Evaluation of Microcontroller Simulation for Transmission Control Units of Passenger Cars

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.
Automotive Software Engineering

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Md Maniruzzaman
Student ID: 398815
Date: 25.09.2018

Supervising Tutor:
Prof. Dr. W. Hardt
TU Chemnitz

Dipl. Ing. Franz Adis,
ZF Friedrichshafen AG

Acknowledgement

I would like to express my sincere gratitude to the people who guided me through the course of this thesis work.

First and foremost, I would like to thank my advisor, Professor Dr. Wolfram Hardt for giving me the opportunity to write my Master thesis under the supervision of his department. I would also like to thank Dr. Ariane Heller for her support and guidelines throughout the duration of the work.

I would like to convey my utmost gratitude to my external supervisor Mr. Michael Sobott for giving the opportunity to work with my thesis under his department at ZF Friedrichshafen AG. His constant support helped me to understand & work further with the topic. I would also like to acknowledge and convey my gratitude to Mr. Franz Adis and Mr. Dejan Jovanovic for their constant help with each and every problem during the work.

Abstract

Transmission Control Unit (TCU) is an essential part of automatic transmission operation in modern vehicles. TCUs control the switching behavior of an automatic transmission system with a control Software, which takes direct input from the sensors as well as from the engine control unit and makes driving a vehicle much optimum and dynamic as well as provides fuel efficiency. Thus, developing intelligent and sophisticated control software for the TCUs is crucial.

To ensure an optimal performance from the TCUs, proper testing of the control software has shown significant results. Software in the Loop (SIL) testing is the one that comes into the consideration when it comes to testing and verification of control software. With the increasing level of complexities in the software modules, the testing of the codes in a real hardware is very complex, risky and expensive. Whereas a SIL simulation promises to provide a much faster and effective testing system without using a real hardware and to improve the development process.

This thesis evaluates the possibility of using a microcontroller simulation for the SIL simulation of control software. The simulations of Transmission Control Unit of vehicles are primarily done by building an executable for PCs. This executable differs from the actual software (hex) that's flashed into the controller. Thus, there are two separate development processes for the simulation and the controller. To solve this, a single simulation of the controller is considered as a solution. But, simulating a whole controller is a very complex and expensive procedure which is not available explicitly. Hence, simulation of the microcontroller is a viable and affordable option at this point of time.

For the purpose of simulation during this thesis work, AURIX TriCore from Infineon Technologies has been chosen as the target controller; which is a high-end microcontroller with 32-bit architecture and preferred in a lot of automotive applications. As for a simulation tool, the Universal Debug Engine from PLS GmbH has been selected for its ability to provide the required criteria. A prototype environment has been imagined and implemented which includes a Program to be flashed on the target controller, simulation of the same code into the simulator and finally, running this simulation results along with the SIL environment. The necessary criterion to run the simulation has been documented herewith.

Keywords: Simulation, Microcontroller, TCU, Multicore, SIL

Content

List of Figures

List of Tables

List of Abbreviations

ECU	Electronic Control Unit
TCU	Transmission Control Unit
SIL	Software-in-the-Loop
MCU	Microcontroller Unit
OS	Operating System
AURIX	Automotive Realtime Integrated NeXt Generation Architecture
GCC	GNU Compiler Collection
ISS	Instruction Set Simulator
SRAM	Static random-access memory
CAN	Controller Area Network
OEM	Original Equipment Manufacturer
RTE	Runtime Environment
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
SW	Software
EXE	Executable
CPU	Central processing unit
FPU	Floating point unit
DSP	Digital signal processing
UART	Universal Asynchronous Receiver/Transmitter
LIN	Local interconnected network
USB	Universal serial bus
ELF	Executable and Linkable Format
HEX	Hexadecimal
TSIM	TriCore Simulator
UDE	Universal Debug Engine
UAD	Universal Access Device
ISA	Instruction Set Architecture
MCONFIG	Memory Configuration
PCONFIG	Peripheral Configuration
ICONFIG	Interrupt Configuration
DLL	Dynamic Link Library
FMI	Functional Mock-up Interface
API	Application Programming Interface

1 Introduction

The number of ECUs in the vehicles has gradually been increasing since its first introduction in 1970 [1] [2]. The number can even be over 100 in luxury vehicles nowadays. This increase can be justified by the increase of added features to the cars to make them comfortable, safe and cost-effective. They are being widely used in Engine control, Powertrain, Brake, suspension, transmission, safety application and entertainment.

An Electronic control unit (ECU) is an electronic device to regulate or control any electrical system of a vehicle. They are integrated in-vehicle systems to automate the functionalities of certain electrical systems. For example, to control certain functions of an Engine, for emergency braking, to cruise a vehicle at a certain speed or to control infotainment systems such as Radio, Navigation etc. [3].

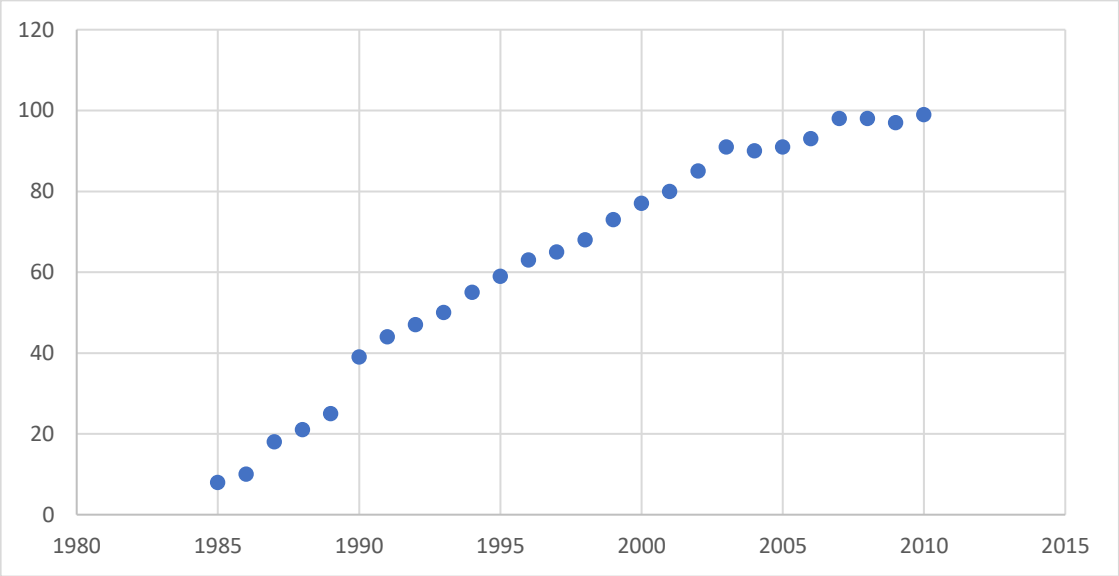

Figure 1.1: Increase in No. of ECUs with years [1]

ECUs can take input data from sensors, bus systems and other ECUs like Engine control unit [4] [3]. ECU uses a closed-loop control system to monitor the output of a certain system using the input from the input sensors. They can gather input data from dozens of sensors and analyses those data within a fraction of a second to perform an operation. To make best out of the sensor data, a sophisticated control software is required to analyze them to make the proper decisions and send them to the actuators. And these actuators are responsible for controlling the desired functions of a vehicle's parts [3].

Table 1.1: Application of ECUs in a car [3]

Application Area	Application
Powertrain & Chassis	Engine Control
	Transmission Control
	Brake control
	Speed assist
Body	Body Monitoring & Control
	Window, Door, Airbag
Safety	Stability Control
	Pre-Crash Safety
	Anti-lock Braking
	Traction Control
Drivers Assistance	Parking Assistance
	Lane keeping
	Cruise control
	Collision avoidance
Infotainment	Navigation, Communication, Radio

Transmission Control Unit (TCU) is one of the widely used ECUs; which is responsible for the proper operation of transmission of a car. The basic principle of a TCU is to take inputs from various sensors, process the sensor data and finally providing outputs to the components of a transmission [5].

Developing a sophisticated TCU is a complex and time-consuming task to do. Using an actual TCU hardware while doing the testing and verification of different hypothetic development concept is expensive and sometimes risky. Thus, simulated test runs are implemented for further improvements.

1.1.1 Motivation

The Motivation behind this Thesis work is to use a simulator for the control software of Transmission Control Unit in Software-in-the-loop testing. It is expected to find the outcomes and benefits if any, of the hex codes into the simulator. The ECU software's are getting more complex more day by day [6]. This is making it difficult to test them in real time while running it on the real ECUs. Thus, the possibility of having unfixed bugs and error in the software is also higher. Besides, potential errors can cause hazards, accidents and thus extra expenses. The problem with the current testing and development process is described with details in the chapter Problem Statement. Software-in-the-Loop testing has several benefits over other testing methods such as Hardware-in-the-loop testing where the presence of the target hardware is essential [7]. The ability of SIL simulation to provide useful feedbacks while running the tests is a major advantage. That provides the Engineers to check and determine the functionalities and efficiency of a System while testing.

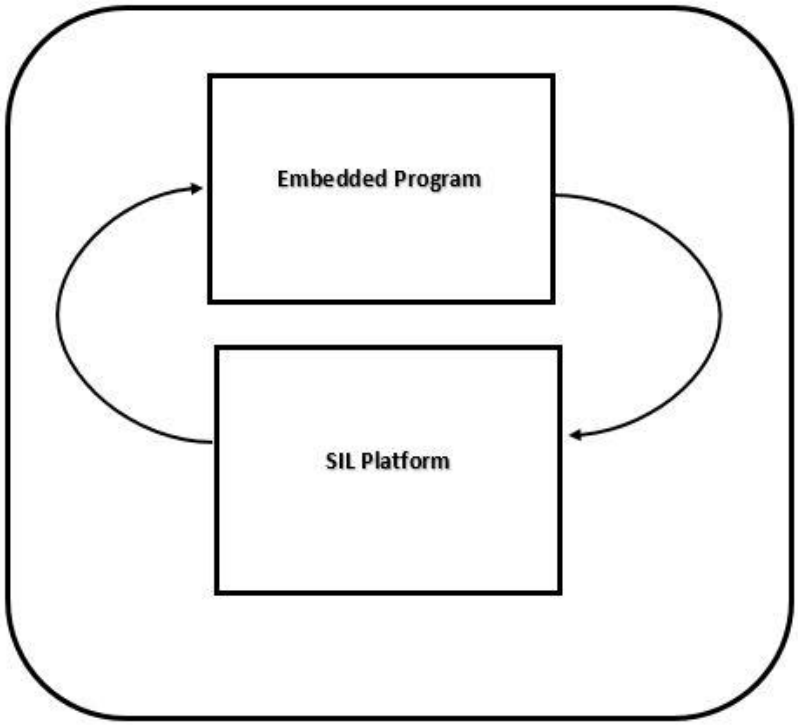

Figure 1.2: Abstract idea of SIL Simulation and testing

The Control Software that controls the behavior of the mechanical component of a system can be tested and verified before they are integrated using simulation. The complex software modules can be managed, system level bugs could be fixed, and overall efficiency could be increased even before the release of the hardware [8]. If

needed, the overall design can be modified or redesigned. SIL simulation is much faster because of not having any real time constraints as in the real hardware. A simulator comes with the facility of debugging, profiling, testing and verification. Therefore, the purpose of a simulator is not only to develop software for a specific target, but also to debug and test already existing control software [9] [10].

Why Use Simulation

A simulator provides an environment with a combination of hardware and software that mimics or imitates certain behavior of a hardware or software product. To ensure quality hardware or software, it is a common practice among the industries to test and verify its functionalities before their production and deployment. Simulation of a circuit before its production gives the ability to test its hazardous and risky behavior and reduce the faulty outcomes [8] [11]. On the other hand, simulation of a software makes it possible to debug the program, edit and modify it and make it optimal for the platform it is targeted to. Analyzing and verifying a theoretical model of a certain product at its early conceptual level is also a big advantage of simulations [12]. A simulator can be beneficial at any level of production of a certain product. Starting from design and conceptualization of hardware up to the point of its production, verification and deployment simulation provides numerous advantages [13] [14].

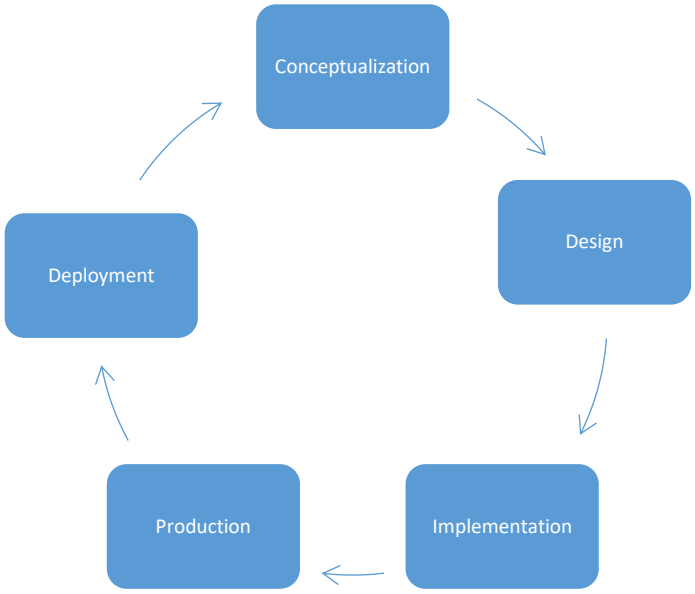

Figure 1.3: Simulation at various stages of a product lifecycle

Importance of ECU simulation

The simulation of ECU simulation can make the development of both ECU software and hardware many times faster. It also makes also cost-effective because the application development can be done long before the hardware is available. The simulation also makes it possible to find errors bugs at the initial stages of large development. The simulation of multiple ECUs can provide the facility to simulate and develop a complete vehicle module. Simulation has also been contributing in the field of “Advanced drivers Assistance Systems (ADAS)”.

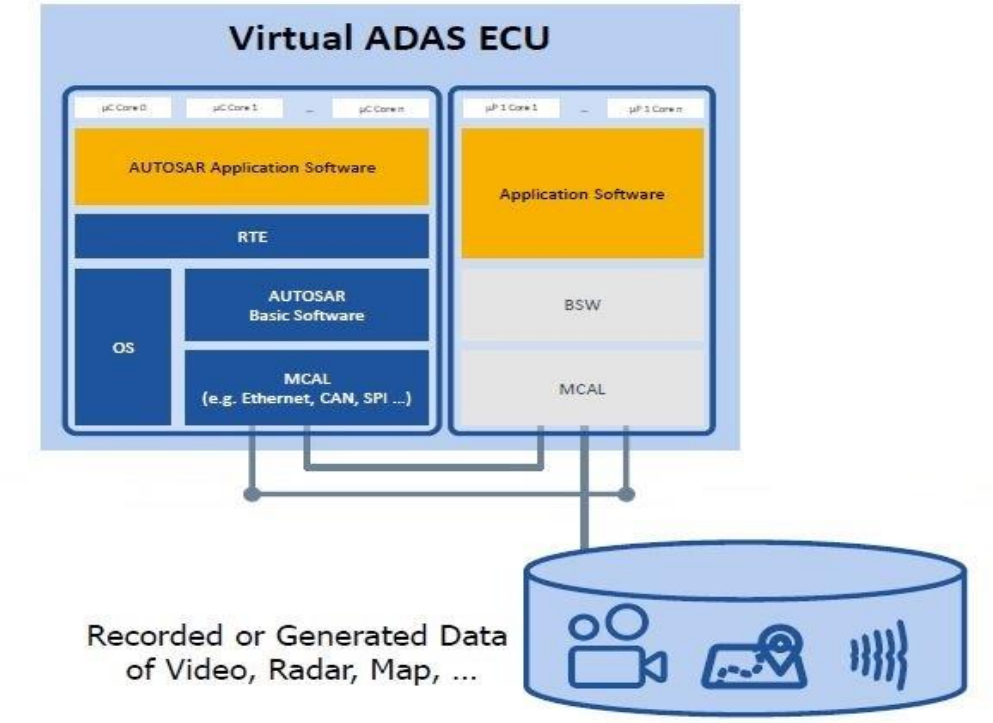

Figure 1.4: Virtual ECUs used in the development of ADAS systems[15]

The parallel simulation of networked ECU with proper communication Bus system has the ability to run and test the function of a complete vehicle model as the Car2X communication. A Virtual ADAS ECU features visualization, simulation of vehicle environment, test automation and the BUS simulation with the communication network to verify and validate efficient ECU software [etas].

1.1.2 Why Microcontroller Simulation?

While the simulation of a TCU would give numerous advantages, the simulation of a complete ECU is a very complex and expensive procedure. In this situation, simulation of a Microcontroller is the choice. The simulation of the target hex code could give a much easier and affordable option to test, debug and analyze the control software that is finally going to be deployed in ECUs. Besides, it is hardly possible to find a simulator for the target TCU that is being used and there's a very limited option of choice for this. On the other hand, there are plenty of microcontroller simulators at this time that is very sophisticated in terms of debugging, testing development of the microcontroller programs. As in the core of a TCU lies a microcontroller, it is a good option to simulate the microcontroller hex codes [3]. As this leaves the possibility of simulating all the peripherals in of TCU, we have to integrate this microcontroller simulation with the SIL platform to test the peripherals behavior [16].

1.1.3 Problem Statements

With an increasing complexity in control software, it is very important to do comprehensive system level testing of the software. Because it is very often that a bug in the system level occurs and it is very expensive to find a fix for these bugs. Therefore, doing a system level testing before prototyping the physical system is very likely to reduce the cost and improve the overall software and hardware performance. To test new functions of the control software, a Software-in-the-loop testing with Softcar tool system is used. With this, verification of change requests regression tests etc. are done.

As shown in the following figure, there are two separate development processes. One of them is for the target and the other is for the SIL test. The behavior of the application software may deviate from the behavior on the target system when tested in Softcar. This causes a major problem.

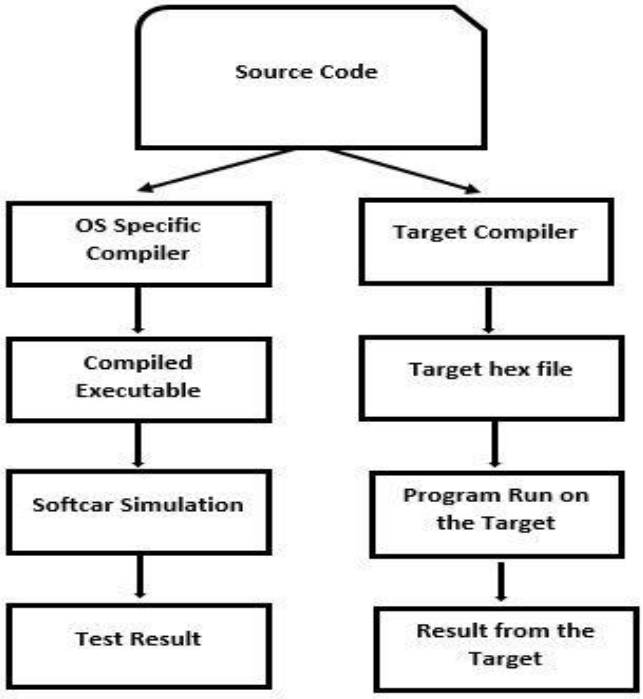

Figure 1.5: Two separate development processes

The application software for the target system is compiled with a target specific compiler; e.g. Tasking compiler or other compilers that are used to compile the programs for the microcontroller. This gives as output an elf file (Executable & Linkable Format). It is an object file that contains all the information including debug information, the hex code, symbol lookups and relocatable table. Whereas, the program for the SIL test is compiled using a GCC, Visual C/C++ or other compilers that creates a win32 executable. This executable is specific to windows platform and deviates vastly from the target code. Thus, the development of application software for target hardware and for the SIL test comprises two separate processes with two separate outcomes. This is the core problem and is needed to be addressed.

An Instruction Set Simulator (ISS) is thought to be a solution at this stage. But, the simulators cannot simulate the peripherals. Therefore, integration with the Softcar is necessary. It is hard to find a suitable and trustworthy ECU simulator that can provide the desired outcomes.

1.1.4 The Goal of the Thesis

The primary goal of this thesis work is to find the necessary criteria to use an instruction set simulator to simulate a specific microcontroller and use this simulation

in a Software-in-the-Loop (SIL) test platform. Simulating a microcontroller can benefit the development of control software on large scale in early stages of the development cycle. The simulator enables to find a lot of early errors and bugs that can be fixed and improved before deploying the program. It also gives a possibility to reuse the target codes among the hardware and software platform. To make use of the MCU simulation in SIL environment the final goal has been divided into few steps. The parts of the goal are as follows:

- Finding a suitable Simulator
- Writing a program for the target hardware and flashing it
- Using the same target code into the simulator
- Run this simulation along with Softcar (SIL platform)

The steps are set in a linear way so that completion of one step would lead to the successful start to the next step.

1.1.5 Structure of the Thesis

This Thesis is structured in nine chapters. The first chapter introduces the scope of the thesis work. The second chapter is about the state of the art. Recent technologies, works, and their fundamentals are presented in this chapter. In Chapter three, fundamentals of the technologies that are related to this work are discussed. Fourth talks about the specific hardware that has been used during this work. In Chapter five, development of concept and methodology, criteria for the prototype environment is described. The proposed prototype and its working principle are described in the sixth chapter. The seventh chapter contains the implementation processes. In chapter eight, the methods for the simulator tool to be integrated with the SIL environment is described. Finally, the thesis ends with a conclusion and suggested future works in ninth chapter.

2 State of the Art

In this chapter, the recent & ongoing work, technologies, and methodologies related to the Electronic Transmission Control Unit and simulation is discussed briefly. The findings related to the abilities and limitations of ECU and microcontroller simulators during the development of a control application are also focused during the research.

2.1 Literature and Product Research

As a part of this Thesis work, a thorough literature research for the concepts of multicore microcontroller and the available simulation technologies for microcontrollers and ECUs have been done. The research for the simulators is divided into two parts- The Microcontroller Simulators and the ECU Simulators.

2.1.1 Simulator for AURIX MCUs

As the primary purpose is to simulate the AURIX TriCore, therefore, finding a specific simulator that supports AURIX and can simulate it is a primary goal.

Table 2.1: Simulator for AURIX microcontroller [39] [40] [41]

Key Features	Universal Debug Engine (PLS)	Lauterbach Trace32	Tasking TriCore Toolset
TriCore Support	Yes	Yes	Yes
Virtual Target	Yes	Yes	Yes
Run Hex file	Yes	Yes	Yes
AUTOSAR support	Yes	Yes	Yes
Interface to compilers	Yes	Yes	Yes
Multicore Simulation	No	No	No
TSim Support	Yes	Yes	Yes
Peripheral simulation	Limited	Limited	Limited

Explanation of the required key features:

- The first and foremost feature that the simulator is supposed to have is the ability to support AURIX TriCore microcontroller. For this reason, among all the available microcontroller simulators, only the AURIX supported simulators are taken into consideration.

- The second function that the simulators are required to have is to support virtual targets to work with. To simulate the functions of a microcontroller program it is essential that a virtual MCU is possible to use instead of a real one.
- The third major feature is to be able to load a compiled microcontroller hex file and run it.
- As the TCU software is developed using the AUTOSAR architecture, it is also important that the simulators support the AUTOSAR architecture.
- Debugging a program during simulation is a crucial thing to do. The program could need modification after debugging. And that would need recompiling the program. Therefore, having interfaces compiler is also a necessary thing.
- The AURIX microcontroller is a multicore controller with three individual cores. Therefore, having multicore support is really important.
- TSim is the simulator variant form Infineon that is specific to the simulation of different generation and model of TriCore controllers. The simulation tools require TSim as core simulator.

Currently, the source-level debuggers from Lauterbach, PLS, and Tasking has the support for TriCore architecture with the ability to simulate the compiled hex code that is targeted to run the AURIX controllers. Along with the on-board debugging using the debugger interfaces, they can also run a virtual target to simulate the instruction set architecture of that specific microcontroller. Support for the AUTOSAR architecture enables them to run and test the real control application on them. Thus, they can contribute to the verification stage of an application. Interfacing with the external processes such as a compiler, IDEs, and API enhances their ability to modify the functions of the application software.

Besides all the features that these simulators offer, they have limited or very less ability to simulate the peripherals. Simulation of necessary periphery modules is also a crucial point during the testing and verification of application software. Because of this limited peripheral support, they are required to interface with external process or software, libraries during the SIL simulation.

The Simulation Model of Periphery modules contains functions and registers that correspond to the physical modules of a microcontroller. It is an extension to the program beside the program that runs the simulated core. For an appropriate simulation of the control application, peripheral simulation is also necessary. The peripheral simulation extends the functional analysis of an application with the

behavioral output of the peripheral devices. The selected simulator tools different approaches for the peripheral simulation.
Lauterbach Trace32 provides several libraries for peripheral simulation. These libraries enable the tool to interface with several external devices.

Table 2.2: Lauterbach peripheral libraries [17]

Library	Function	Description
LPC2xxx Timer Model	Interfaces with LPC2xxx timer module from Philips	Contains the source code of timer module, general model, and scripts
Vectored Interrupt Controller	Interfaces with Vectored Interrupt Controller	VIC model source, model definition, dialogue for CAN devices interrupts
MPC565 Interrupt Controller Model	Interface with MPC565 controllers from NXP	interrupt controller code, controller definition, model header & source
NEC v850 Interrupt Controller	Interface with NEC V850 from Renesas	models compiled for win32, dialogue and initialization scripts
CortexM3 Nested Vectored Interrupt	Interface with Cortex M3 controllers	Nested vector source, model and definition for compiler

The Tasking TriCore toolset and the Universal debug engine make use of the Peripheral control processor of the AURIX Microcontroller. The peripheral control processor (PCP) is a special-purpose processor used to control peripheral units of AURIX microcontroller. A PCP controller is a combination of General-Purpose Registers, instruction pipeline, arithmetic logic unit (ALU) along with a control and status registers & logic. The instruction set of PCP is optimized for the purpose of the set of operations that it has to perform.

The Universal Debug Engine extends the configuration of a Target core to support the peripheral configuration with the help of a PCP controller. UDE comes with PCP Assembler support that visualizes the functions at runtime, supports multicore debug, scripts and testing functions.

The Tasking TriCore toolset uses a specific PCP simulator configuration for peripherals. The standard TriCore Instruction Set simulator runs the simulation of the

core instruction architecture. Then, it initializes the PCP plugin simulator. This set up is done in the manual configuration file.

2.1.2 Simulator for ECUs

Using an ECU simulation in the SIL environment is also a potential option. For this reason, a study on the available ECU simulators is also done in order to determine the possibilities of using them. An ECU simulator can simulate the functionalities of a control unit that is not available or not yet manufactured. They can simulate multiple configurations of ECUs and also emulate the BUS communication between ECUs.

Table 2.3: ECU Simulators [42] [8] [43]

Key Features	Silver (QTronic)	VECU (GAIO)	ISOLAR-EVE (ETAS)
TriCore Support	Yes	Yes	Yes
Virtual Target	Yes	Yes	Yes
Rapid Prototyping	Yes	Yes	Yes
AUTOSAR Support	Yes	Yes	Yes
Run hex File	Yes	Yes	Yes
Multicore Simulation	No	No	No

The simulators in the table above are ECU simulators in general. Just like the microcontroller simulators, they are expected to have the features like TriCore support, virtual target, AUTOSAR support, and multicore simulation facility. Unfortunately, none of the simulators can simulate multicore processors of the microcontroller.

Silver (QTronic)

Silver provides a virtualized version of an ECUs and creates a virtual development platform for the software developers to develop control software without the need of an actual ECU. It creates a closed-loop co-simulation platform with complex models to develop control applications. With a virtual ECU in this platform, the production parameter can be set, measurement and calibration with CCP/XCP-based tools such as Inca, CANape or CALDesk is also possible [18].

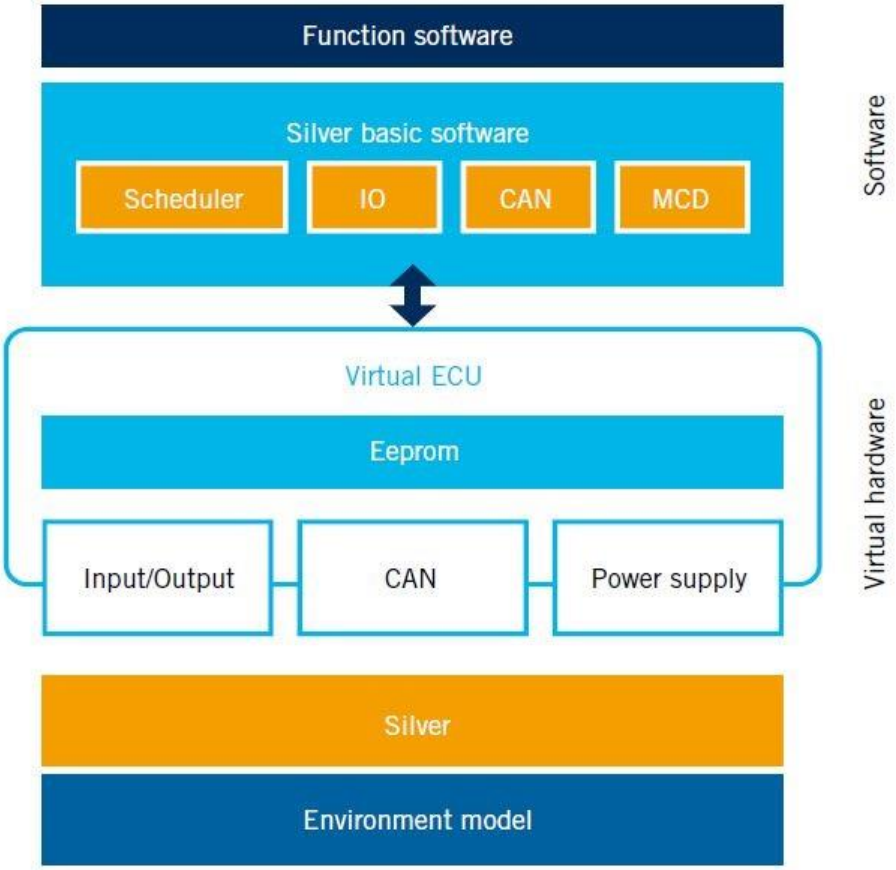

Figure 2.1: Virtual ECU in Basic Software in SIL simulation [19]

Silver Basic Software libraries are the basis of a virtual ECU. The functions from these libraries are used to replace function calls of the function of real control software and the hardware aspects of an actual ECU. Using this, the distinction between configuration and the use cases are drawn. The basic software can be configured during the runtime. This makes it possible to significantly influence the behavior of a virtual ECU to run without the need for recompilation. The run-time configurable nature of the ECUs provides a longer life expectancy than the ones that are generated off-line. It also reduces the necessary communication between development partners [18].

VECU (GAIO)

Virtual ECU (VECU) from GAIO is a virtual verification platform for ECU Software with an Instruction set simulator. VECU uses a Simulator based Processor in the loop (PIL) simulation to simulate an ECU. VECUs are capable of Performing Software

Testing of an ECU in the early stages of the development cycle by simulating the Target Code.

VECU-G includes a source level debugger which makes them capable of debugging an application with breakpoints and step by step execution. The instruction simulator also has an interface to connect with MATLAB/Simulink models of the vehicle system [20].

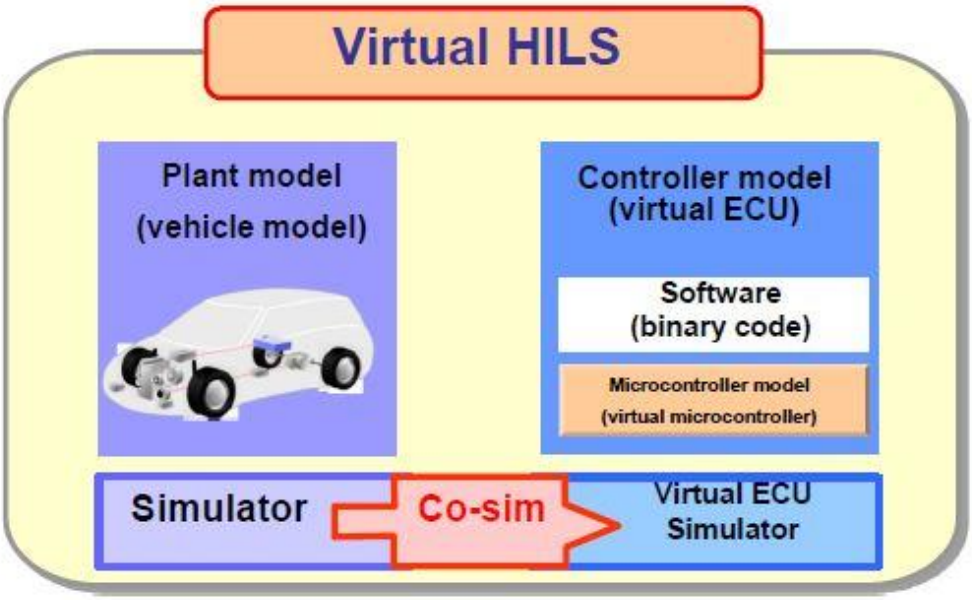

Figure 2.2: A Complete simulation with Hardware, Software and Plant model [20]

Besides SIL simulation, VECU can also provide a virtual HIL platform that can work with vehicle model and simulated control software. VECU uses a co-simulation to connect between the vehicle model and the controller model. The binary code for the target model is then simulated to examine the behavior of the vehicle model.

ISOLAR-EVE (ETAS)

ISOLAR-EVE from GmbH provides a virtual platform for running and simulating ECU Software many times faster than on the real ECUs. Both Black-Box and White-Box testing can be done to ensure that the functionality of an application meets the specified requirements. It is possible to run multiple ECUs parallel to each other. It provides a fast and continuous integration feature. ISOLAR is very flexible in terms of timing ECU software content and Interfaces with another system. It means, they can run virtual ECUs at a desired speed faster than or slower than the real hardware. It is also possible to run a complete or part of an AUTOSAR or non-AUTOSAR basic Software from any provider [21].

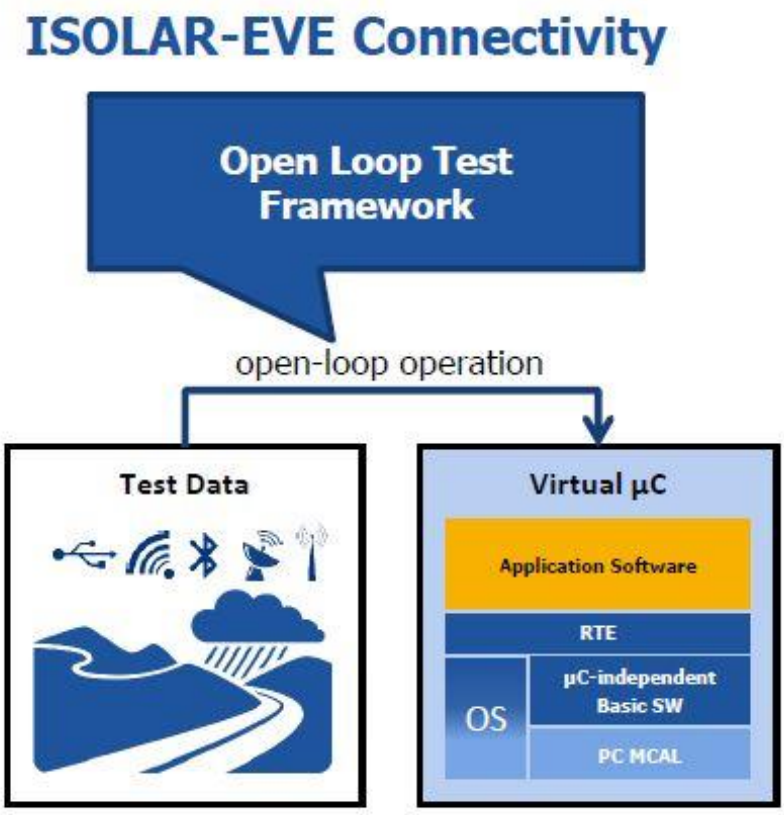

Figure 2.3: Virtual Microcontroller in Isolar-Eve [21]

Isolar also works with Functional Mockup Interface (FMI) and Simulink models and can also establish CAN connection with CANoe simulation tool from Vector Informatik. Included test framework eases the automated testing.

The studies on the simulation technologies and the methods have helped widely in developing a concept for the prototype environment.

3 Fundamentals

This chapter focuses on the fundamental of the technologies related to the automotive transmission system, transmission control methods and the simulation of TCU. The simulation of microcontroller carried out during this work aims to improve the process of developing and test the application software of a Transmission Control Unit (TCU). The TCU is a specific type of Electronic control unit (ECU) that controls the behaviors of an Automatic transmission. An ECU generally has a core processor, input and output modules.

Table 3.1: Main Components of an ECU [3]

Type	Element
Core	Microcontroller
Memory	SRAM
	EEPROM
	Flash
Inputs	Analog input
	Digital input
	Supply Voltage
Outputs	Logic Outputs
	Relay Drivers
	Injector Drivers
	H-Bridge Drivers
Communication Links	Housing

The microcontroller at the core works as the computing device for the ECU. The input module is responsible for taking an input signal from the ambient and sensors. And after the necessary computation from the microcontroller, the output module provides the necessary output signals to the hardware components that are planned to be controlled by the ECU.

ECUs can take a large number of parameters and process them to make optimal performance in operation of various hardware systems. They take a number of electric signals transmitted from different sensors. Then they process all those data to generate control signals for the actuators for the hardware. The control program

with a sophisticated control algorithm. is loaded in the memory that makes all the calculations and decisions. The program is executed by a microcontroller.

3.1 Transmission Control

The automatic transmission has been gaining its popularity because of the effortless shifting facility while driving the vehicles. The fundamental difference between a manual transmission and automatic transmission lies in the principle to control the input power from the Engine shaft and transferring it to the output e.g. wheels [5] [22] [23].

In a Manual Transmission, the engine power is controlled by two shafts in a manual transmission system.

- Input shaft: takes power from the engine
- Output shaft: delivers the engine power to the wheels

The shafts are coupled with gears. This mechanism can have different forms and ratios. In a normal case, a manual transmission has 5 forward gears and a reverse gear. To shift from one gear to another, the load on the gear has to be removed so that there’s no transmission while changing the gears [24] [25].

On the other hand, an automatic transmission can shift gears of a car automatically without using a manual clutch and provides a vehicle the necessary speed and torque while it runs [22] [26].

Figure 3.1: ZF 8-Speed automatic transmission [27]

The two main types of automatic transmission are i) Hydraulic automatic transmissions/ Stepped transmissions and ii) Continuously variable transmissions. Automatic Transmission systems usually contain hydrodynamic torque converter, multi-gear systems and shifting elements [28]. Thus, the typical elements are-

- planetary gear set,
- torque converter/ dual clutch arrangement,
- transmission pump and
- Hydraulic controls etc.

Automatic transmissions can transmit engine power using several methods. The torque converter couples the engine's flywheel to the transmission using a fluid coupling. The power splitting in planetary gear sets and transfer gear sets gives automatic transmissions a higher torque capacity. The planetary gear set uses the fluid and friction to function. It has a Sun gear in the center of the gear set and a few planet gears that rotate around the sun gear. The Planet gears are connected by a planet carrier. There’s a ring gear that engages the planet gears. The planetary gears take the signal from clutches and brakes to decide which components of a gear system move and which does not. Thus, the gear ratios can be changed according to the needs. Automatic transmissions typically use multiple planetary gear sets coupled in the gear set. The shifting elements take a big portion of the whole transmission system [28] [5] [26] [29].

Modern automatic transmission systems use a shifting method known as a clutch to clutch shifting. In this method, one shifting element can be engaged while others are disengaged simultaneously. This method makes the shifting process easier without disrupting the power flow. For this reason, an automatic transmission can change gear without interrupting the engines power flow [28] [16] [25].

Table 3.2: Comparison of manual & Automatic transmission [21]

Manual Transmission	Automatic Transmission
• Uses Gear Pair	• Uses Planetary gear
• They use Clutch pack to distribute power	• Uses a torque converter to distribute power
• Use of a manual clutch to shift gears	• Automatic adjustment of gears depending on the speed
• Manual control over the transmission system	• Automatic control by the vehicle itself
• Less convenient to drive	• Easier and convenient to drive
• More cost effective	• Expensive than manual
• Low maintenance	• High maintenance

Because of the convenient nature of automatic transmission people are making it a preference over the manual ones. An automatic transmission needs less attention for the shifting of gears and allows to pay more attention to the road. Thus, it makes it driving a vehicle lot easier [23].

Besides manual and automatic transmission systems, there are also some other mixed type of transmission systems used vehicles. Automated manual transmission, Dual Clutch transmission, continuously variable transmission, and Hybrid transmissions are the example of these types. Automated manual transmissions are the manual type of transmissions in which the process of gear shifting is automated. In a dual clutch transmission system, it takes the advantages of both manual and automatic transmissions. Therefore, it is highly efficient with manual control and has the power shifting ability without interruption. In continuously variable transmission the speed ratio is varied continuously without interrupting the power flow. And in case the hybrid transmission system, there are two power sources. One is from the typical internal combustion engine and other is the electric source [28] [30] [31].

Electronic Transmission Control Unit (TCU)

Since its introduction in the 1980s by Renault and BMW, the electronic transmission control unit has gained significant popularity. In recent years, the number of electronic transmission control has the biggest hike in America and Japan, which is around 80% and a little less in Europe. A TCU is an electronic device that takes sensor data and analyzes them to control the automatic transmission system. The transmission control Unit is divided into two main parts- hardware and software [31] [32].

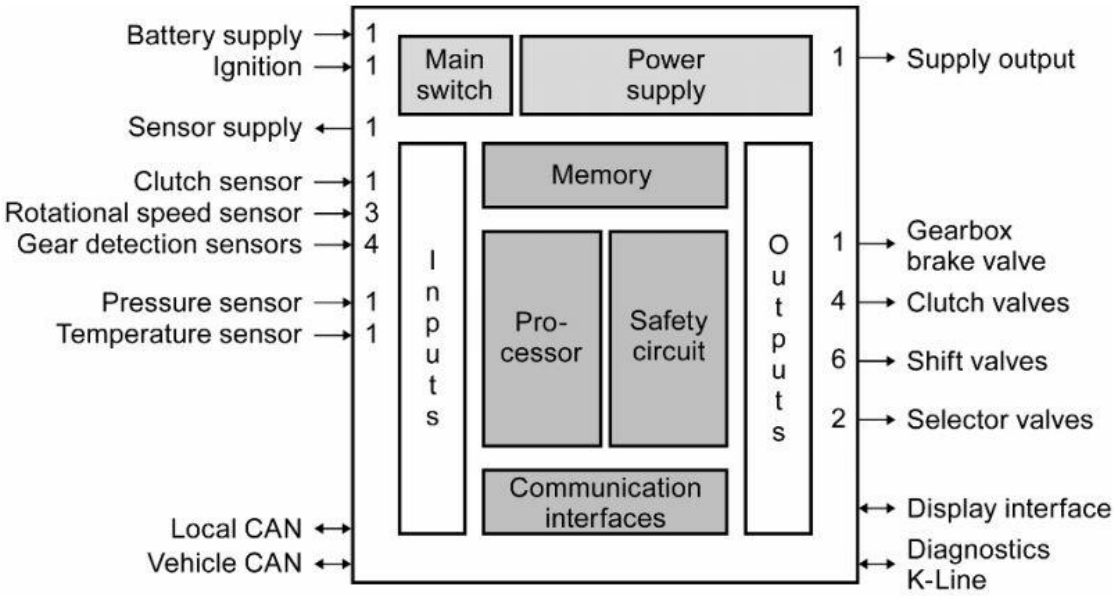

Figure 3.2: Hardware design of a TCU [26]

A TCU is a combination of electrics, electronics and hydraulic components along with sensors and actuators. The hardware contains electronic components- microcontrollers as a processing unit, different sensors, input signal circuits, Safety circuit, interfaces for BUS communication, power supply, peripherals, safety circuit, and output modules. The erasable programmable read-only memory (EPROM) stores the required program and data. The Microcontroller is used as a core computer with a control software to evaluate input signals from the sensors & other components, computation of algorithms and to provide output signals to the actuators.

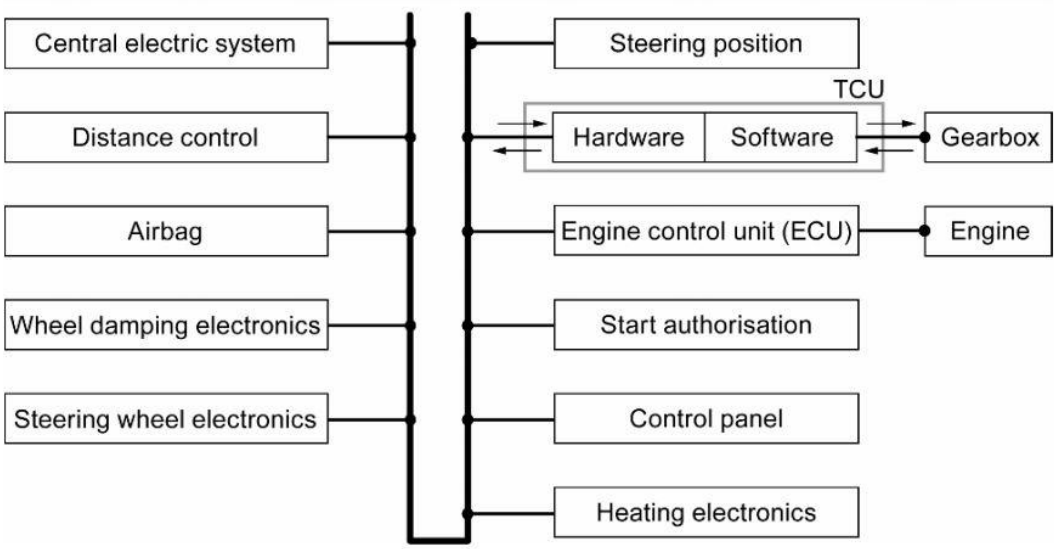

Figure 3.3: Communication of the transmission control unit (TCU) with engine control and other systems in the vehicle [26]

The operation of a TCU involves acquiring a signal from engine sensors, automatic transmission sensors, and the signals from other electronic controllers like the engine control unit (ECU). The TCU then decides by evaluating the data to change gears of the vehicle for optimum performance. And doing so it makes the vehicle more fuel economy and reduces emissions.

A communication bus system is necessary to establish networking among the components in a vehicle. Controller Area Network (CAN) bus is generally used to establish communication with the system and with other ECUs in a network. For the safety purpose of the control unit, a watchdog circuit is used.

TCU Software

The software part of the TCU works as the decision maker and brain. It has two major components. Application software and Data needed to be analyzed. A TCU software has three major functions-

- Vehicle functions
- Basic functions and
- Hardware-related functions

Vehicle functions encompass the behavior of a vehicle in total. Basic functions are the functions that are required to operate the transmission system. And the hardware related functions are the functions that are similar to an operating system which are required to operate the control system itself [33].

TCU software is developed according to AUTOSAR architecture. AUTOSAR (AUTomotive Open System ARchitecture) is a standardized ECU software architecture for automotive industries. AUTOSAR aims to improve complexity management of integrated E/E architectures through increased reuse and exchangeability of SW modules between OEMs and suppliers [34] [35].
The architecture includes four main layers.

- Application layer

Contains the application functions, primarily model based

- Runtime Environment (RTE)

Abstraction of the TCU hardware

- Basic software

Basic services for communication, input, output, memory and system functionality

- ECU Hardware/ Microcontroller

The requirement analysis of a TCU software shows the major functional areas that should be considered during the design of a control Software. In this case, the first category is the one with the functional requirements. Functional requirements are the ones that are involved in the interactions between the transmission system's functional components and environments [25]. For example, how an automatic transmission with different components should function and communicate, how the data collection and processing is done. Thus, the data collected from different signals and processing of all these signals comes as a first functional requirement. The signals come from the analysis of the velocity of the vehicle, the position of the accelerator pedal, the hand lever, brake signal etc. After this, the decision making of auto-shifting is done by the control algorithm.

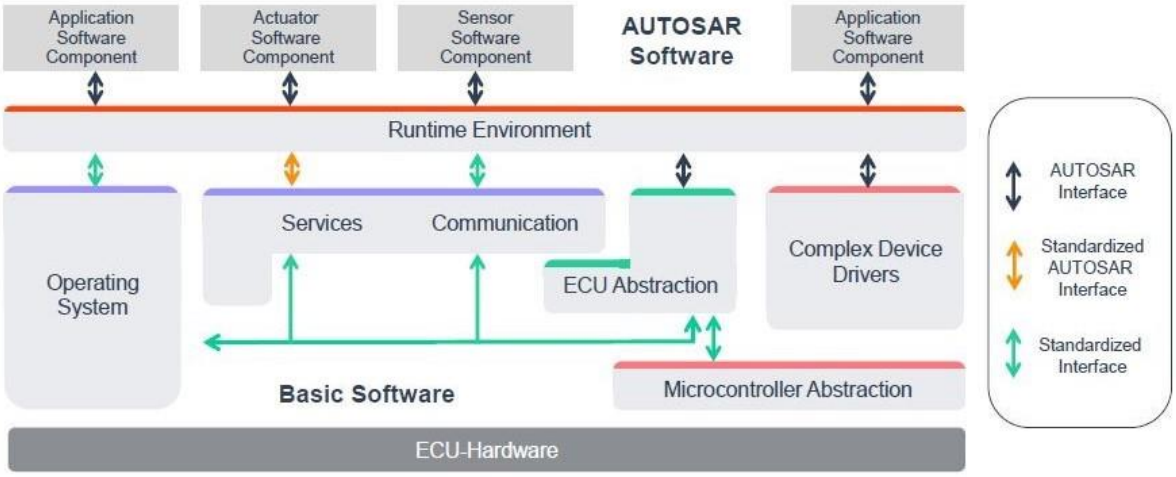

Figure 3.4: AUTOSAR architecture [27]

AUTOSAR provides different functional blocks for the integration of the application layer with the microcontroller. The growing complexity of the Electrical and electronic components has led to the requirement of a standard software architecture that can provide a generalized way of integrating all these components [35] [36].

3.2 Development Model for TCU Software

V-Model

The V-Model stands for Verification & Validation model. It is widely used by Automotive OEM and manufacturers. For the purpose of developing and manufacturing Automotive ECU's, V-model is followed. The left part of the shape V stands for verification while the right part stands for validation stages. It possesses some similarities with traditional Waterfall model. The V-model works with a series of linear and sequential phases within its life-cycle. The advantage of V-model over traditional waterfall model is the testing and feedback system in each individual phase of the development life cycle.

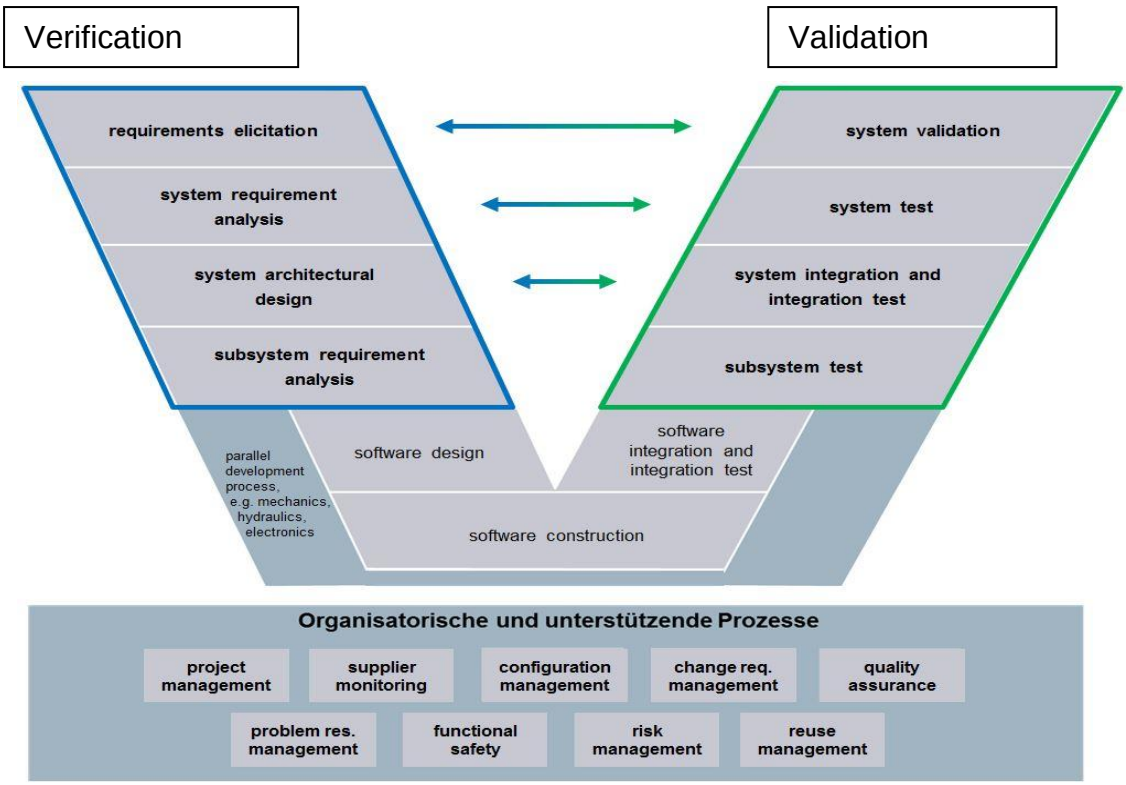

Figure 3.5: V-Model of Development

3.3 ECU Simulation

The ECU software for contains thousands of parameters and signals from different vehicle parts and sensors. To obtain the expected outcome from the software these parameters are needed to be calibrated, tuned and examined. Doing these tasks on the real vehicle system using ECUs is a very time consuming and expensive task. This is why a computer-based ECU simulator is used to test, calibrate, and tune different virtual parameters. This simulation has the potential to automate and speed up all these processes [12].

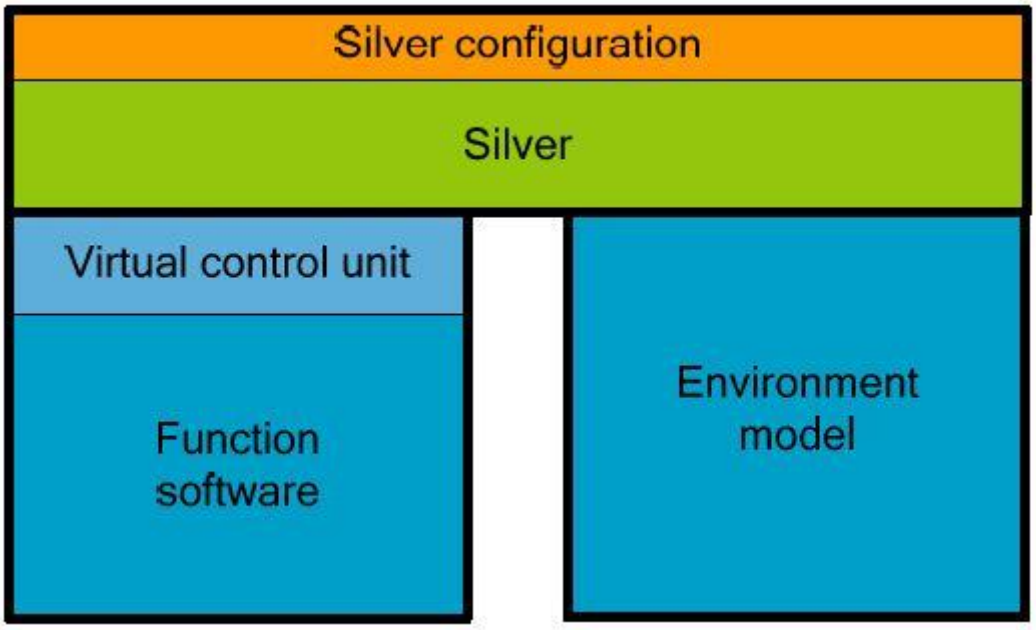

Figure 3.6: Virtual ECU in SIL simulation [12]

A simulated control unit can simulate the relevant functions of an actual electronic control unit. It can connect the function requests of the control software which is made to the BIOS with the corresponding input and output signals of the environment model. Afterward, the software can read sensor values and set the outputs.

.

3.4 Simulation Technologies for Microcontrollers

The simulation of microcontrollers is widely used at present both in scientific researches as well as by the engineers. Engineers are using the simulators to virtualize, prototyping and firmware development even before the actual microcontrollers are coming to real field [37] [38]. With the various purpose of simulating microcontrollers, different types of the simulator are being used at present. These simulators feature the ability to simulate various microcontrollers starting from smallest 8-bit architecture to 32-bit multicore complex microcontrollers [39]. There are a lot of free simulators and a number of commercial paid simulators as well.

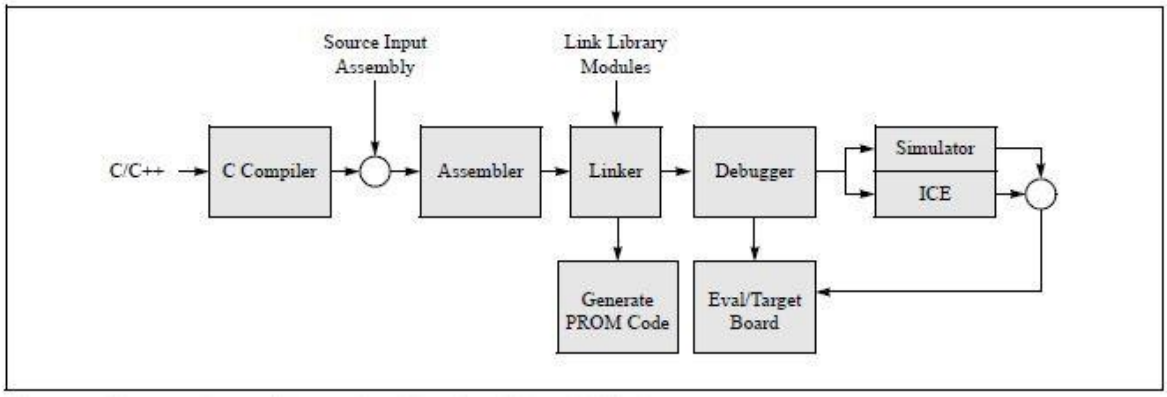

Figure 3.7: Simulation toolchain [33]

Conventionally, Microcontroller simulators are used in design, development and analysis phase of control software or a firmware. The simulation applied on the schematic enables to write and apply software both for the processor architecture and peripherals. It is possible to interact with the design using displays, buttons, indicators etc. [38].

The Major applications of microcontroller simulator are spread in-

- Design
- Simulation of processor
- Analysis
- Debugging and Diagnostics

Using a simulation enables to provide feedbacks during the planning and designing phase before manufacturing a real-world system. This way, the correctness of a system can be evaluated, and modifications can be made to the design with ease. Simulators can also be used to demonstrate a system and also to provide training on the specific systems.

Instruction Set Simulation for TriCore

The Instruction Set Simulation (ISS) is one of the most important steps in developing an efficient processor architecture and application software. It is a simulation model that is typically written in lower level programming languages which is a simple and fast way to mimic an embedded processor and develop efficient software for it. They are used for the development of application software for the processors that are not explicitly available yet [40] [41].

They usually come with following features-

- A released Library for specific processor architecture
- Middleware Library
- Graphical User Interface (GUI)
- Debugger for each processor family
- Load binaries (hex or elf)
- Instance of processor cores
- Script languages support for the testing purpose

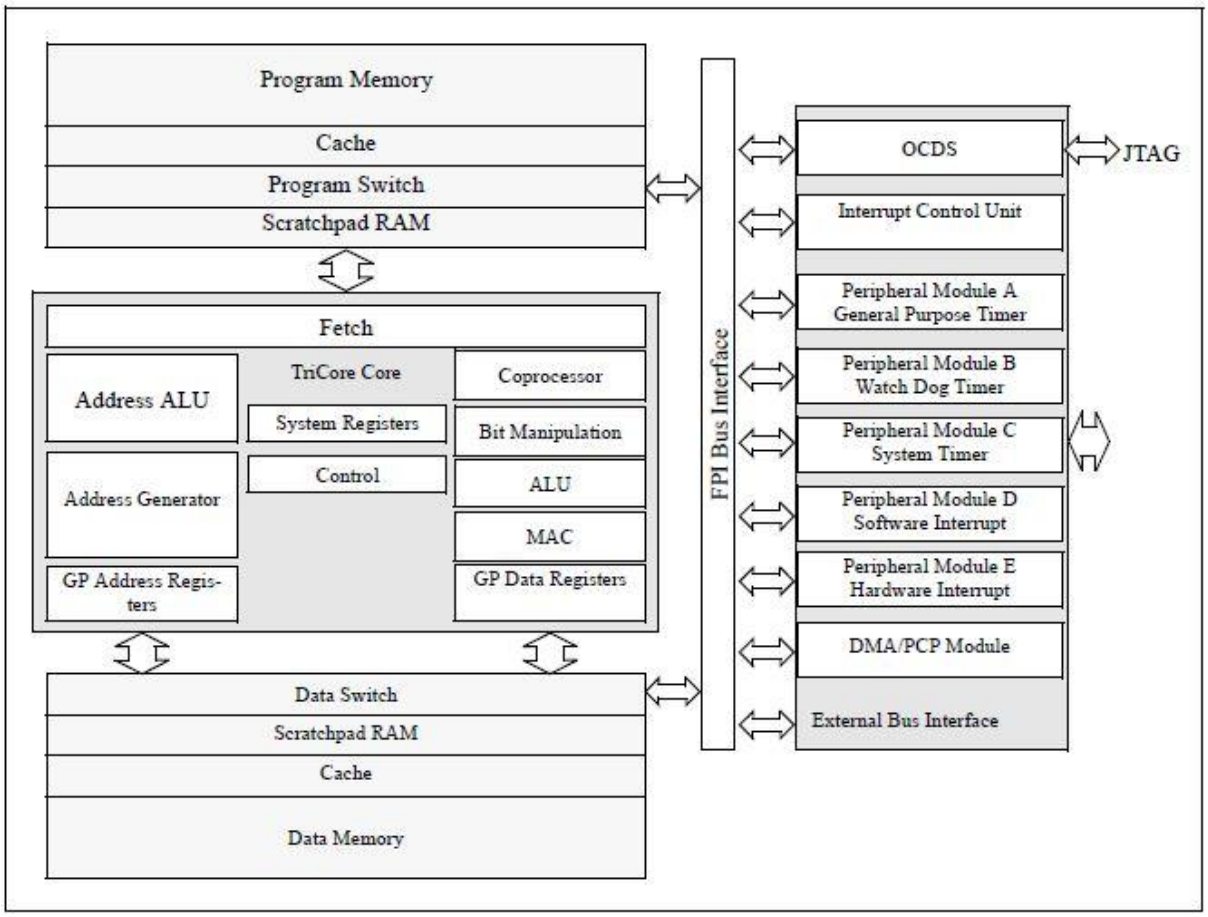

Figure 3.8: Simulation Model of a TriCore based Controller Chip [33]

With an instruction set simulator, development and debugging of an application program for a specific processor model can be done with ease.

3.5 Software in the Loop (SIL)

Software-in-the-loop (SIL) is an essential and integral part of the development process of control Software. It is a test method where executable codes for a certain

mechatronics system is tested with a model environment. SIL is a closed loop testing system which enables the developer to debug and analyze the program with faster changes [13]. With SIL separate versions of a control software can be run in a closed loop simulation on computers using simulated hardware components. The control software is wrapped with a wrapper that emulates the function of a specific hardware component [42] [43]. And the wrapper itself is a part of the SIL which run on an operating system.

A SIL system can include the following features and functionalities-

- Simulation
- Debugging
- Automated Test
- Code Coverage
- Measurement

During the early stages of design and development, SIL makes it possible to find out design flaws and bugs in the program. This results in a more optimized and mature control software. There are no real time constraints during SIL simulation. Whereas in other testing such HIL requires explicit real-time requirements. A SIL system can run many times faster than a real hardware emulation. This facilitates them to be very agile. Required changes can be made frequently without delays. SIL is also less expensive compared to HIL systems [43].

Softcar

Softcar is an in-house SIL platform and simulation tool for ZF that has been developed by the developers of ZF itself. The platform along with Softcar includes other components such as common memory, model, compiler etc.

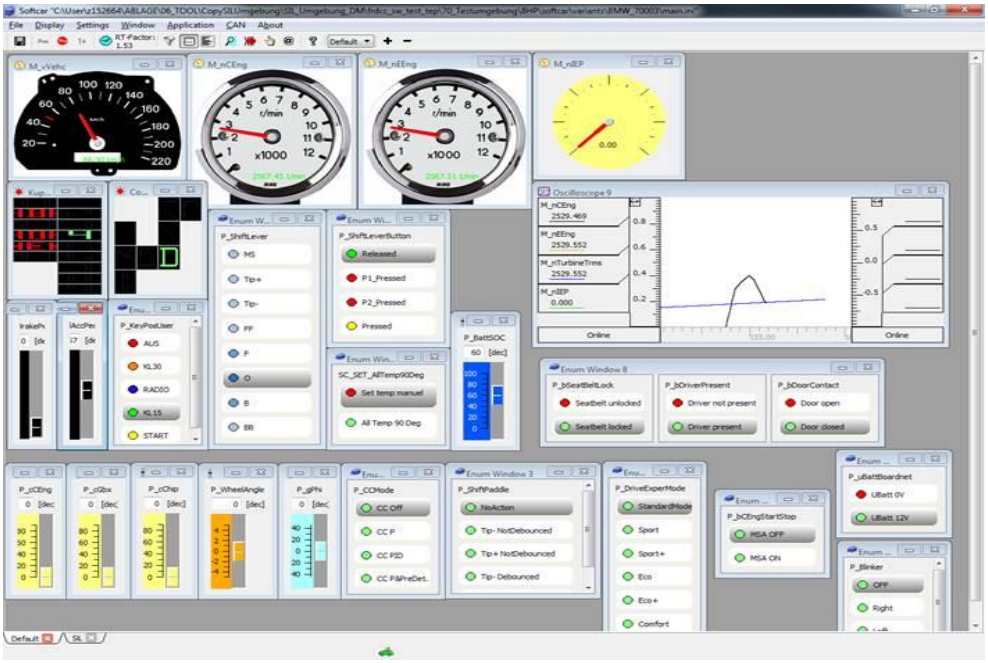

Figure 3.9: Softcar user interface [44]

Softcar-functions are implemented into two groups of processes:

- Internal processes, which are the same overall projects (main functionality of Softcar)
- External processes for project specific functions (model, ECU-SW)

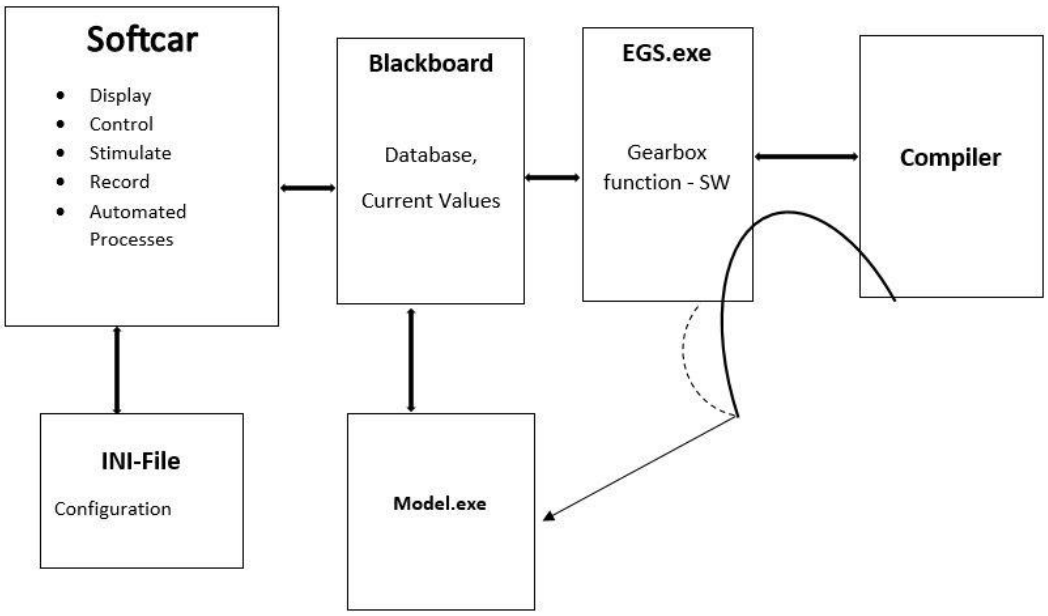

Figure 3.10: SIL with Softcar [38]

- Softcar.exe works as the simulation tool and loads the internal and external processes to run the whole SIL platform
- Config file (INI) is the initialization file that contains the initial instructions and is used at the startup of the Softcar process.
- Blackboard is the common shared memory that is used by the individual programs to communicate among them and with the process. All these programs are registered to the Softcar access.
- EGS.exe is the executable with the gear functions that has to be simulated during. Softcar can load other executables as well in order to simulate their functionalities.
- Model.exe is a Model of the target system
- Compiler – There are a lot of compiler options to be used such as Borland, GCC, GNU, Visual C or other

Advantages of Softcar on Windows platform:

- Unlimited Code size
- Unlimited Data size
- Debugging in the IDE of VC.Net or BC5.02
- Less name conflicts between Softcar modules and ECU- respectively model-modules
- Softcar stays active (display, controlling) when ECU-SW is stopped in the debugger.
- Critical errors (general violation of protection...) can be matched quicker.
- Only 1 config-file

Softcar is separated into two windows:

1. The control panel for controlling and
2. The output-window for display.

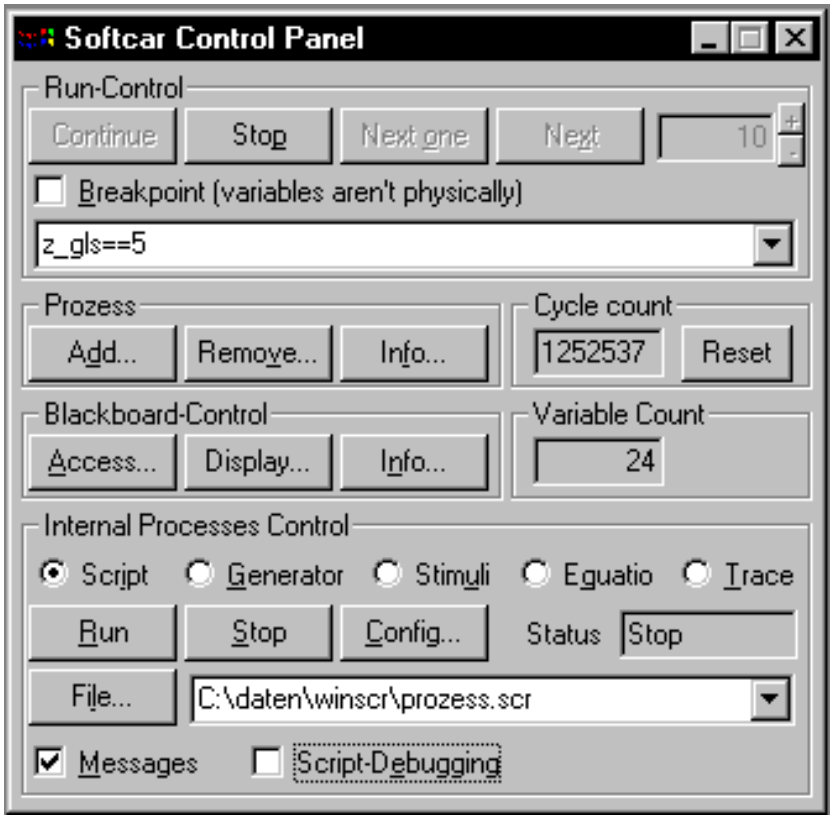

Figure 3.11: Softcar Control panel

As shown in the figure, the control panel controls an overall process. The internal functions are controlled through this control panel. It is always in the foreground and divided into four sections. Starting a process, continuing or stopping it and stepping through several steps lies in run-control section. Breakpoints can also be set via the control panel. Processes can be added or remove from Process option. Blackboard-Control lets the configuration of the common memory. The blackboard works as kind of database that holds all current values and their setups. For the internal process control, Softcar has an integrated scripting facility which is very advantageous.

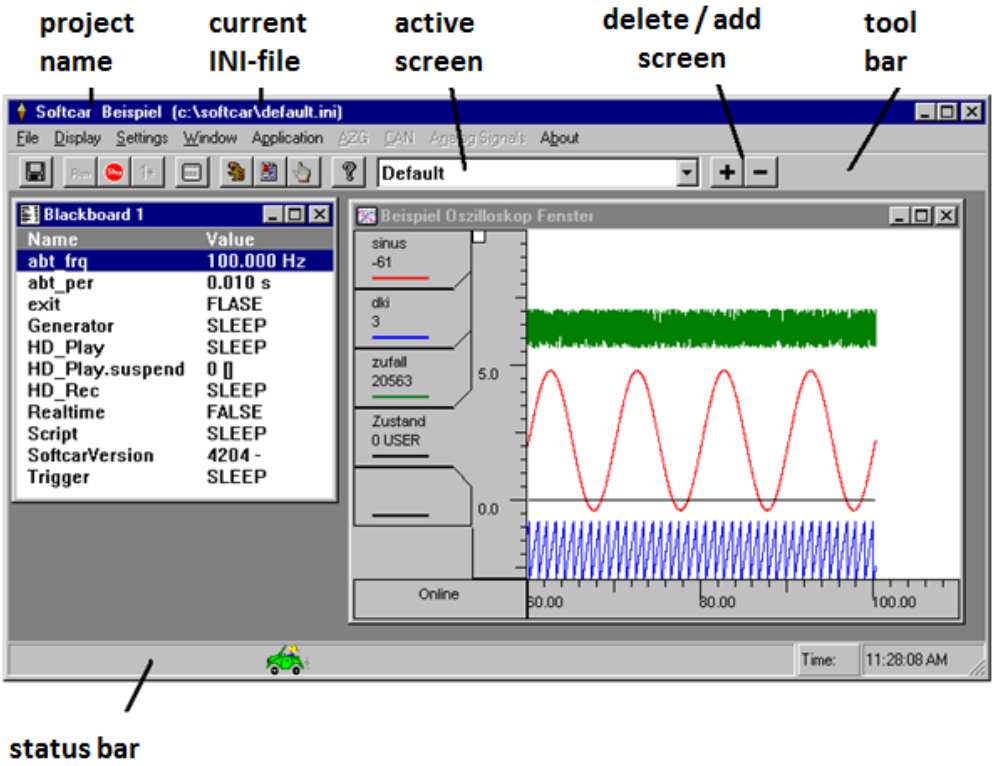

Figure 3.12: Softcar output window [44]

Softcar output window shows the result of a simulation process as a text output or as oscilloscope graphs. The status of different process parameters can also be seen in different fields.

The Scripting language in Softcar serves for the automation of test sequences. The script language is an interpreted language, which is compiled and executed during runtime. Repeatability and reproducibility of the standardized test at any time make testing much more efficient. Using the script language, it is also possible to combine and connection of all Softcar-provided options without the presence of a user. A debug file is generated during the execution of a script file. This debug file contains all executed commands and problems occurred while the Softcar process is run. In this way, the logical mistakes within a program of script commands can be found.

Softcar uses XCP for calibration and measurement. XCP stands for Universal Measurement and Calibration Protocol. It is a universal protocol that is used to connect ECUs with calibration and measurement systems which enable read & write access to the microcontroller memory at runtime. This protocol does not depend on the type of network that is used.

4 Target Hardware and Technologies

4.1 AURIX TriCore

The full form of AURIX is Automotive “Real-time Integrated NeXt Generation Architecture”. It is an extremely high-performance microcontroller family developed and manufactured by Infineon Technologies. The first generation AURIX has a 32-bit multicore DSP architecture with three independent cores. Their main application lies in the automotive industries in the field of safety and higher performances. AURIX is primarily used in Powertrain technologies, Safety and chassis domain control systems. They provide a significant increase in performance [45] [46].

The microcontroller that has been used for the purpose of this thesis work is AURIX TC275 model. It belongs to the first generation of AURIX family. It has three processor cores; two of them are optimized for high performance. With a clock frequency of 200MHz, they can execute three instructions within a single clock cycle.

Table 4.1: AURIX TriCore Specifications [45]

CPU Core	Cores	3
	Max. Frequency	200 MHz
	FPU	Yes
Program Flash	Size	4 Mbyte
Data Flash	Size	384 Kbyte
Cache	Instruction (P/E)	16 Kbyte/ 8Kbyte
	Data (P/E)	8 Kbyte/ -
SRAM	Size TC1.6P	120 Kbyte
	Size TC1.6E	112 Kbyte
	Size LMU	32 Kbyte

The third core is efficient to exchange data with the peripherals with maximum one instruction per clock cycle. It consumes the least power among all the cores. All these processor cores are connected by a crossbar that runs at a full CPU speed. It is a high-end microcontroller which is highly sophisticated powerful features in the market [37] [39].

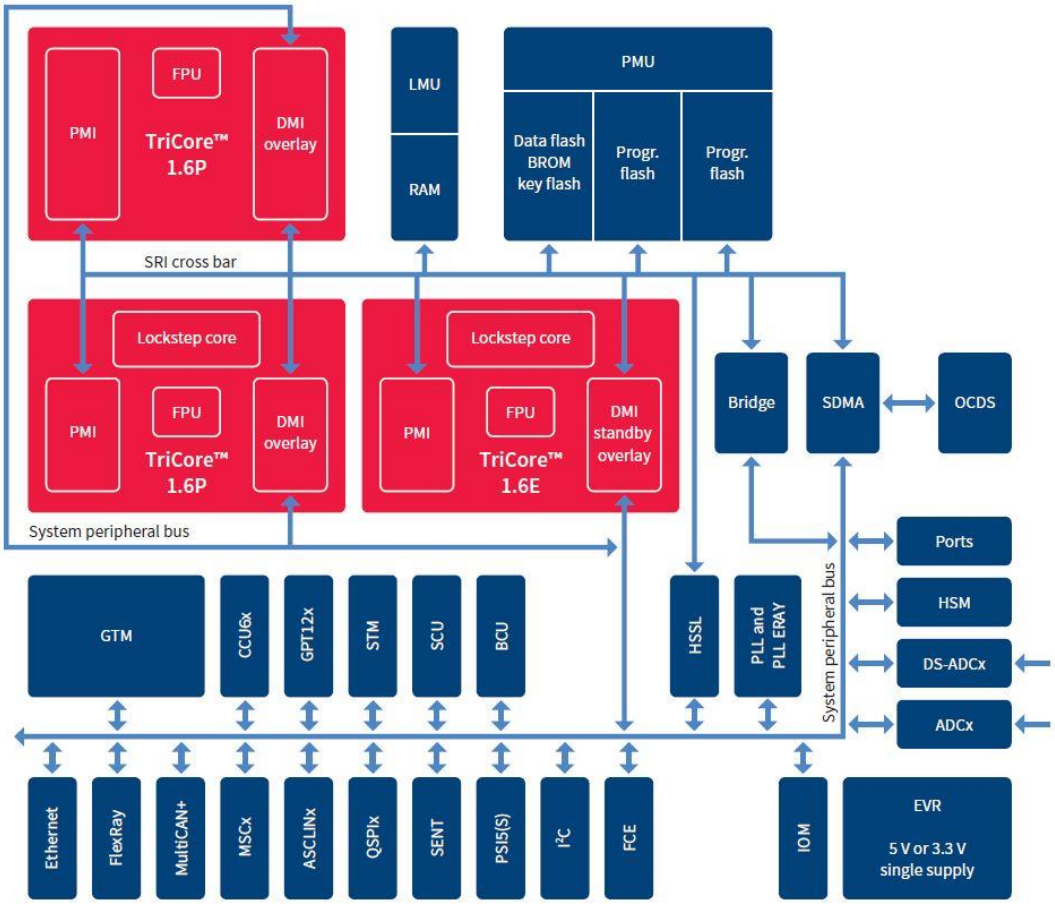

Figure 4.1: AURIX TriCore Architecture [44]

TC275 series comes with three powerful features, making it able to provide higher power, speed cost-effective embedded applications. The features are:

- Reduced Instruction Set Computing (RISC) processor architecture
- Digital Signal Processing (DSP) operations and addressing modes
- On-chip memories and peripherals

The DSP architecture provides the microcontroller with computational power to efficiently analyze real-world signals and make a decision after complex calculations, while RISC provides high computational bandwidth. The RISC load/store architecture provides high computational bandwidth and On-chip memory and peripherals support the high-bandwidth, real-time embedded control systems. The on-chip Debug support gives higher visibility and controllability of control software and system under hard real-time constraints [45].

4.2 TriBoard TC2X5 Evaluation Board

TriBoard TC2X5 is an evaluation board from Infineon Technologies that provides access to the features and capabilities of AURIX TriCore's powerful architecture. It enables development of TriCore applications with respective tools. It comes with a variety of configurations of memory and peripherals to interface with the environment. It also provides an on-chip debugging interface. The TriBoard comes with a number of features:

- Infineon's TC2X5 Controller in LQFP-176 Package
- FlexRay Transceivers
- Safety device (optional)
- High Speed CAN Transceivers
- USB to UART bridge
- Ethernet Gigabit PHY
- Serial EEPROM
- LIN Transceiver
- Crystal 20MHz (default) or External Clock
- USB miniWiggler JDS for easy debugging
- 8 Low Power Status LEDs
- 8-DIP switches for configuration
- access to all pins of the controller
- 100mm x 160mm (EURO-Board)
- optional power supply via USB

The microcontroller within the TriBoard requires two to three different voltages to be supplied. These voltages are provided internally via the microcontrollers supply TLE (+5V; +3,3V; +1,3V) or via the microcontroller itself (+3,3V; +1,3V) [47]. If a stable voltage is supplied, then it can cause power-on reset within a short period. With the button press, a manual power-on reset can be executed [47]. The board has to be connected via a power supply connector or via a USB cable.

The board has 14-19 LEDs, a 20 MHz clock assembled which can be replaced, serial connection to PC, miniWiggler debugger interface and serial EEPROM. For communication using BUS within a network, it has MultiCAN, LIN, FlexRay and

Ethernet. There is also a safety device which is connected when it is required to have safety option [47].

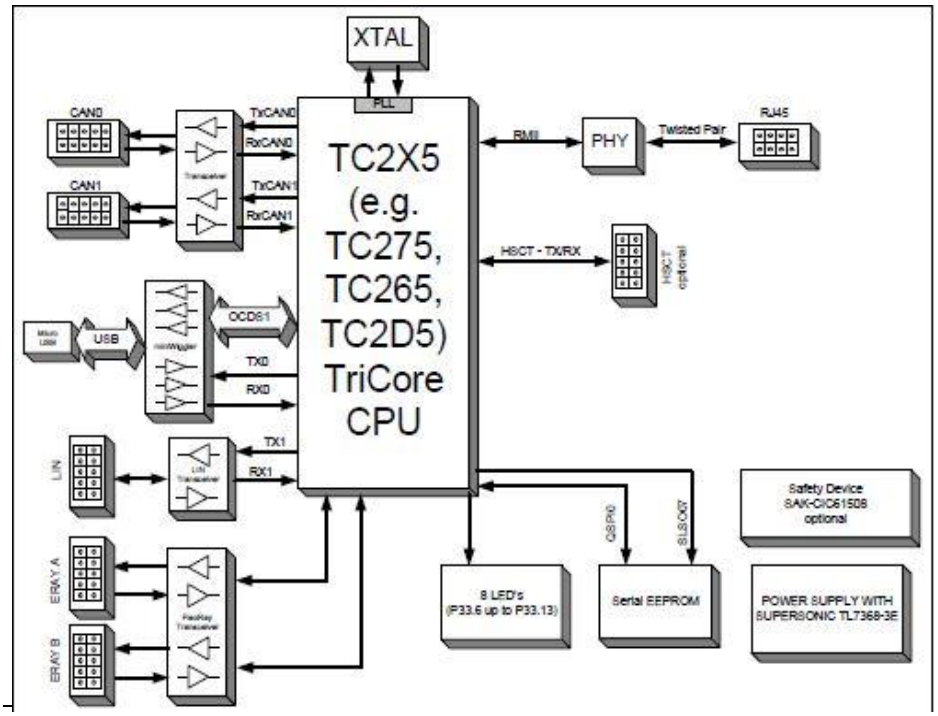

Figure 4.2: TriBoard Block Schematic [44]

TriBoard comes with several development tools that let easy development of the TriCore application. These tools also allow downloading the application to test and debug with corresponding tools. The board has interfaces to connect & communicate with CAN, FlexRay, Ethernet, LIN, and miniWiggler.

5 Concept and Methodology

The concept building of solving the existing problem starts with the Incorporation of an instruction set simulator (ISS) into the Softcar system for the respective target system. So that it would be possible to determine the deviations in the behavior through comparative test runs and then examine them in a more targeted manner. If there are no deviations, it can be assumed that the changes made do not conflict with the selected toolchain.

5.1 Concept Development

While developing the concept for the whole work, several key points were noted at the beginning. The procedure then followed those points and successful completion of each step was counted as a milestone for the work. Here the key points are described briefly:

- Finding a Suitable Simulator

The first challenge is to find a suitable simulator that can run a simulation of the specific hardware that has been introduced for this thesis work; in this case AURIX TriCore TC275. Extensive research has been done for this purpose. Starting from an ECU simulator, the possible features of all available Microcontroller simulators has been investigated. Several demo simulators have been requested from the companies and after a trial installation, their functionalities and potential advantages over each other have been studied. The main focus while making a preference of the simulator has to be either on the ability to simulate the Transmission Control Unit with peripherals or the ability to simulate the specific target microcontroller model.

- Writing program for the microcontroller

After the selection of a suitable simulator, the first step would be to develop a sample program for the target microcontroller. The program has to meet some specific criteria so that verification of the simulation result could be done by following those criteria. Writing a program for the microcontroller would need planning and requirement analysis of the intended program.

- Simulation of Microcontroller hex code using an Instruction Set Simulator

After the successful program execution on the target board, the next step would be to simulate the same program on the selected simulator. The program would be

compiled to an ELF file and this same compile object would be flashed into the hardware and simulated on the simulator as well.

- Instead of two separate executables use of a single controller simulation
  As described before the program would be compiled into an ELF file and the same object file would be used to flash the microcontroller as well as to simulate.

- Analysis & Defining the Criteria to have a single development & testing process for the Target. The process of using the same object file has to be documented and which challenging criteria come during the process has to be listed. For this, the functionalities of the simulator, the SIL environment and the development process need to be observed with care.

- Reuse of target HEX codes
  One of the crucial parts would be to reuse the target hex code for the development, simulation, and testing. In the current procedure, the control software is compiled using different compilers when using in the simulation and to flash the program on the Transmission Control Unit. The program should be planned in such a way that a single compiler will be used to compile it and then flash it on the MCU. The same compiled code will then be imported into the simulator to simulate and debug it.

- Integrate simulation result with Softcar/ other possible SIL System
  The final step would be to connect the microcontroller simulation with the SIL platform so that it can be run within the cycle and tested.

5.2 Criteria to be met

The introduced Prototype has to meet the following Criteria:

- Run and debug the target code on the simulator
- Instruction set simulation
- Multicore Simulation
- Peripheral Simulation
- Connection of microcontroller simulator with SIL platform

Run and debug the target code on the simulator

Among all the criteria that the prototype has to meet, the first criteria is to run the microcontroller code on the simulator. The code is first compiled for the real hardware and flashed into the target to check out the functionalities. After that, the same compiled code has to be taken into the Instructor set Simulator. The debugging and testing of the code has to done using simulator breakpoints and stepping through.

Instruction set simulation

Instruction set simulation is important for the simulating the behavior of an application for a programmable controller architecture. It mimics the behavior of a specific processor by taking the instructions and setting the register values according to the instructions taken. In this way, the machine codes are read a full system simulation is done instruction by instruction. The ISS simulation is faster than the interpretive simulation.

Therefore, it is important that the simulator that has to be used has the ability to do the instruction set simulation capability. In this case the ability to simulate the instructions or codes that are targeted for AURIX TriCore processor architecture.

Multicore Simulation

The AURIX microcontroller used for this work has a multicore architecture with three independent cores. Each core has its own RAM and memory to run a specific function that is meant for that core. For this reason, the simulation environment has to have the power to simulate the multicore architecture of a microcontroller. Simulating multiple cores is a very complex and expensive procedure to implement. But it would be very effective if a multicore simulator for the AURIX could be used.

Peripheral Simulation

Besides the simulation of the processor core, it also very important to simulate the peripherals. Because with peripheral simulation, the real benefit of an application simulation can be attained. The behavior of the peripherals that are meant to be controlled with the microcontroller is required for the verification and validation of a control software. The conventional microcontroller simulators has limited or very few peripheral simulation abilities. Which limits the testing of an application with the peripherals that are required. The Softcar tool used by ZF has a wide range of peripheral simulation ability. Therefore, it is required to connect and run the microcontroller simulation with Softcar. So that along with the processor simulation,

the peripheral simulation can be done to make the full system simulation and test a complete application.

The connection of microcontroller simulator with SIL platform

The last criteria is to connect the Microcontroller simulator with the SIL platform. For the purpose of peripheral simulation and the testing of the application, the simulation has to be done in a loop with other elements of the SIL platform. To fulfill this purpose, the connectivity of the microcontroller simulator and the SIL platform has to be analyzed to establish a method for the necessary communication.

5.3 Simulation Tool Selection

To make a decision and select a suitable simulator from the list of found simulators, the following criterions are taken into consideration-

- Support for the AURIX microcontroller
- Support for the multicore architecture
- Simulation of AURIX
- Development and Debugging features
- Previous experience of the team with the tools
- Availability and License cost

For the simulation of specific AURIX microcontrollers, there is a simulator tool from Infineon Technologies. It is called TSIM (TriCore SIMulator). The TSIM simulator facilitates the features to simulate multiple versions of AURIX microcontrollers. Infineon has worked with a few third-party development partners to develop a suite of development tools which are known as source level debugger. These source level debuggers come with features like integrated development environment, debugging simulation etc. [48].

Lauterbach T32, Universal Debug Engine & Tasking TriCore VX tools wrap the TSIM simulator with a debugging interface. These debuggers feature both onboard debugging and a simulation [42]. These three tools are somewhat similar in using the TSIM simulator as a core for instruction set simulation. All these tools are tried out with demo licenses. After analyzing their functionalities & features, considering their availability and license in the team the Universal Debug Engine (UDE) from “PLS Programmierbare Logik & Systeme GmbH” has been selected.

The UDE comes with an eclipsed based development environment, TSIM as a simulator and Universal Access Device (UAD) to connect the tools and the target hardware with the host pc.

In this chapter, the prototype visualization and the process of implementing the prototype is discussed. The necessary criteria that has to be met are described.

6 Prototype Environment

6.1 Prototype Visualization

The proposed prototype environment includes several components. First of them would be a Debugging tool with TSim simulator. Then this tool has to be coupled with the SIL environment to run it in a loop simultaneously. The complete process should include-

1. Writing a program for the AURIX and flashing it to the microcontroller using TriBoard
2. Taking the compiled program as ELF into the simulator and simulate and verify the core behavior

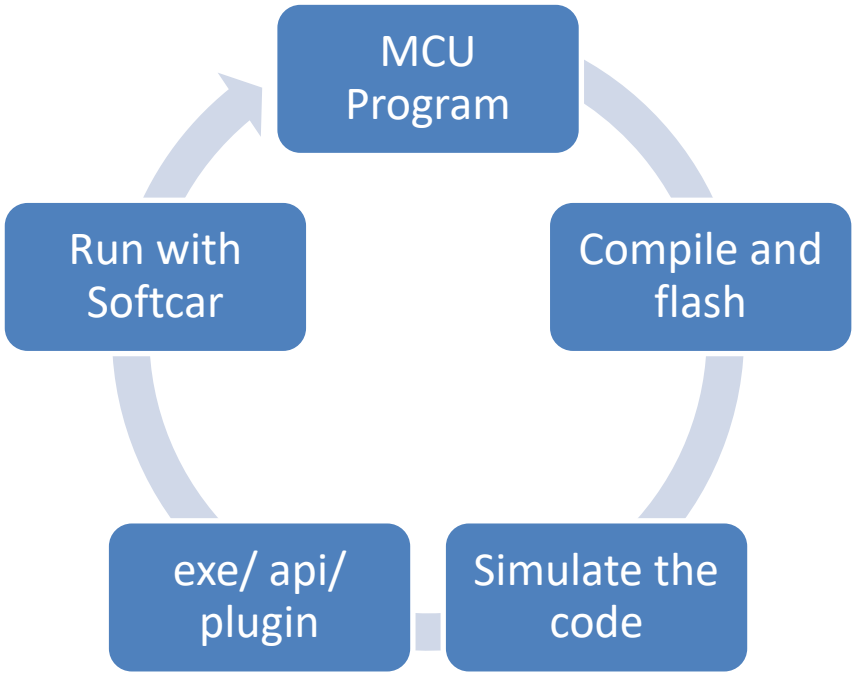

Figure 6.1: Steps for the prototype development

3. After successful simulation, running the TriCore simulation along with Softcar in the SIL process
4. For this reason, a connection mechanism has to be defined that can run the MCU simulation and the Softcar platform in parallel for data acquisition.

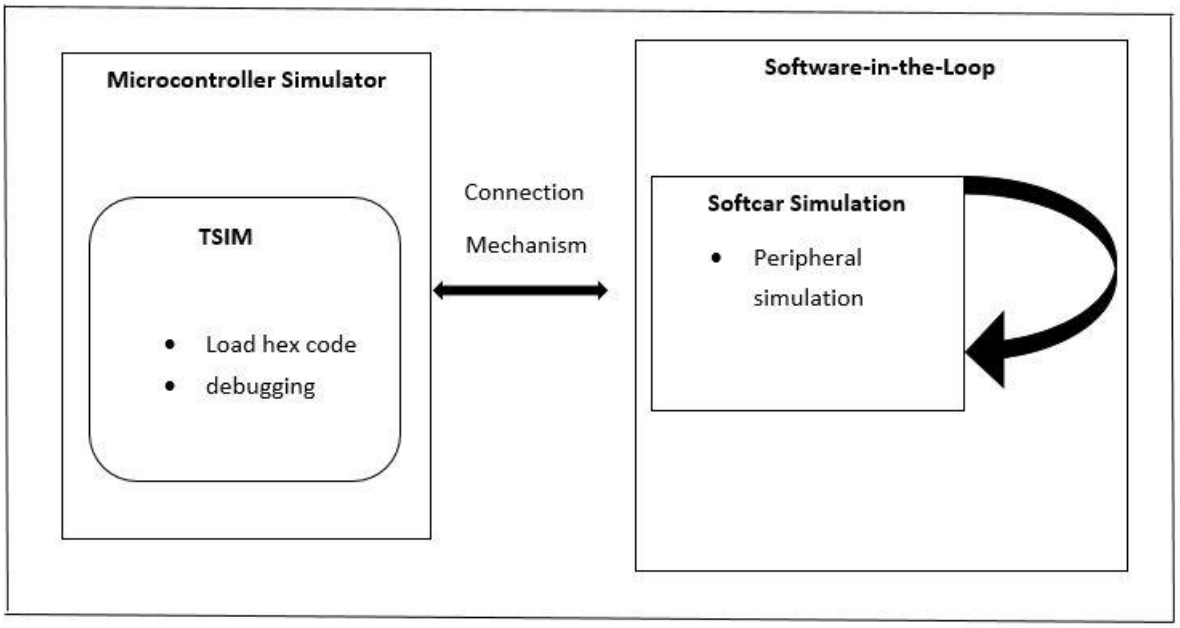

Figure 6.2: Structure of the Prototype

As the figure depicts, the prototype environment encapsulates the SIL platform and the MCU simulator in one complete process. Both the processes have to run in parallel to each other and transfer required data and commands in order to keep the process in a loop.

The simulation of microcontroller should be able to take commands from Softcar in order to stop or continue with a process that is running in the loop. The SIL platform gives the possibility to show the output of the peripherals that are not explicitly available on the microcontroller simulators.

6.2 Implementation Method

- **Program**

  The program for the target microcontroller started with the decision on the simulation tool. For the development of the code, the Eclipsed based integrated development environment with Universal Debug Engine (UDE) is used. Development of a program for on this environment is convenient because of the prebuild and configurable configuration for the AURIX microcontroller. After writing the program, it is compiled and then flashed to

the target board using Universal Access Device that comes with UDE to program the microcontrollers. The UDE environment has the ability to create projects with both actual Target configuration and a simulated target configuration. The compiled hex code is then, loaded into the simulation program.

- **Simulation**
  After creating the simulation project, the same compiled hex code is imported into this project to do the simulation.

- **Integration with SIL environment**
  For this purpose, extensive studies are carried out to find the possibilities. The communication protocols supported by the simulator and the SIL platform is analyzed.

7 Implementation

7.1 Program for the target microcontroller

The programs for the microcontroller, for example, can have an analogue input signal and a digital or analogue output signal that can be controlled through the analogue input or vice-versa. A Microcontroller with multiple processor cores gives not only performance boost as a hardware but also gives a significant performance improvement in software side. Having three cores makes the AURIX controller to work optimally even without functioning at the maximum clock frequency. The developers can pick an optimal frequency for a certain application to make it consume less power.

While a multicore processor gives a lot of performance benefits, it also makes it a challenge for the software developers to write a program that utilizes the power of it properly and make it perform as desired. To benefit from this and exploit the potential of a multicore architecture proper programming techniques has to be implemented. When a program runs on a single core, several tasks share it, on the other hand for a multicore processor the tasks have to be designed to run on separate cores concurrently. Doing so would make it beneficial to use a multicore processor.

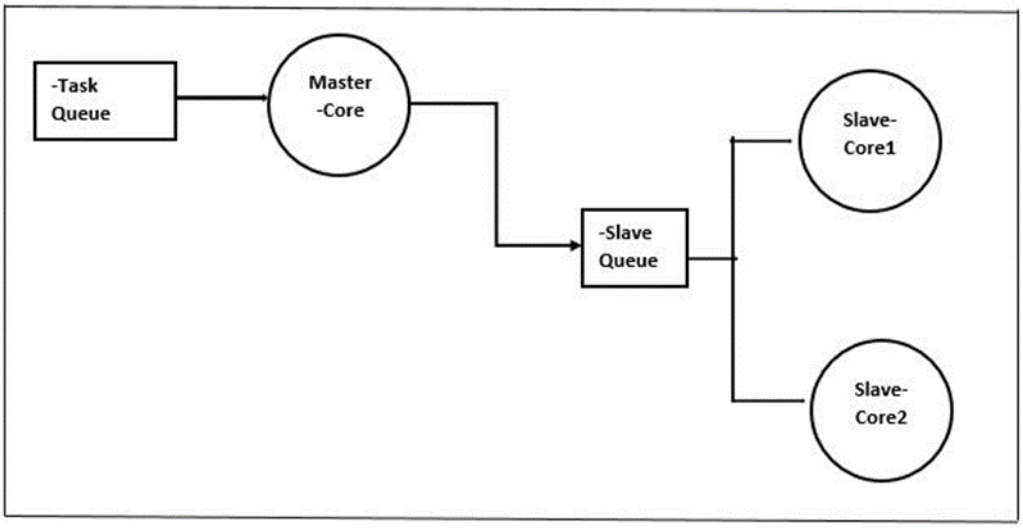

Figure 7.1: Task distribution in Master-Slave model of multicore architecture [46]

To distribute several tasks to the multiple cores or threads several techniques are known to exist. Task Parallelism is one of them. During the design of a program task parallelism and a suitable processing model has to be identified. There are two models that are used mostly. Master/Slave model and the Data Flow model.

In the master-Slave model, one master core is responsible for controlling other cores work assignments. This core schedules and allocates the tasks that has to be executed on the other (slave) cores [49].

Writing a program for the AURIX microcontroller requires planning & designing to make use of its multicore architecture. Writing a multicore program for a pc is a bit convenient in the sense that the operating system manipulates a lot of scheduling and decision making. But in case of a microcontroller, the burden is on the programmer. Two of the cores of AURIX are performance cores which are equipped with Lockstep cores and the third core is known as Efficiency core. Each individual core has its own RAM with a global and local address.

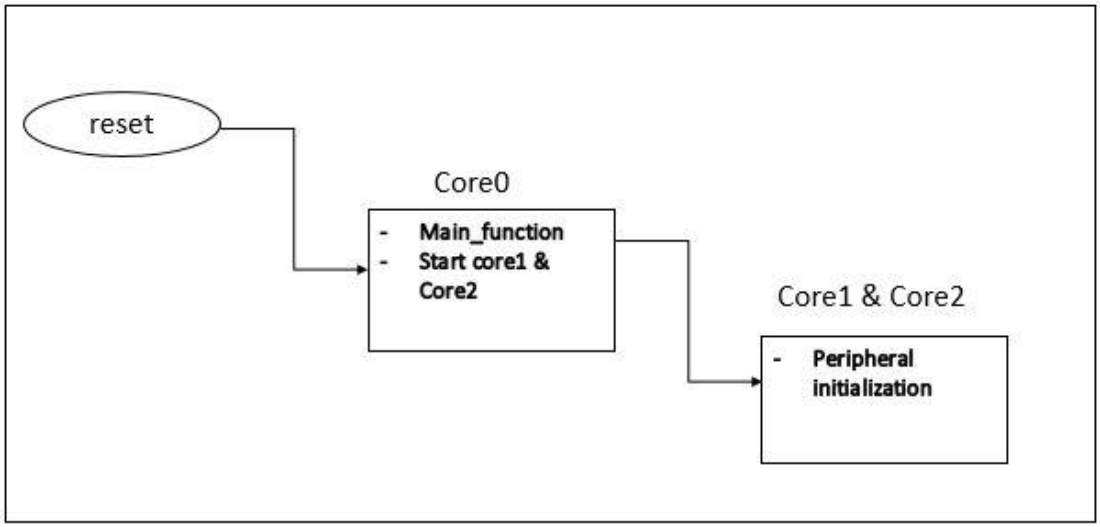

Figure 7.2: Program initialization after each reset

There is a global ROM that is shared by all cores. The start addresses of the cores have to be reconfigured for the startup phase so that all the cores refrain from starting at the same address and run the same functions. Each core can be started with its unique ID and inside the main function, they can be switched with a switch-statement. Thus, it can be decided which core is going to be executing which part of the program. After the reset action, typically the CPU0 is started to keep only one entry point of the program. This core initializes the peripherals and starts the other cores- CPU1 & CPU2. At this point, CPU1 and CPU2 start executing the respective startup and program code [50] [51] [52].

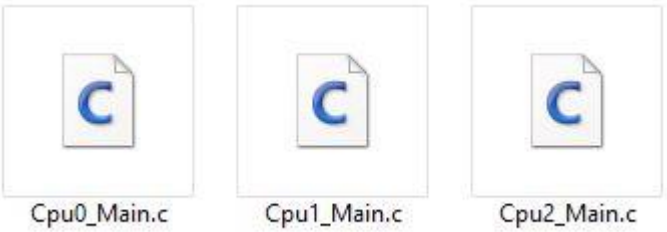

Figure 7.3: Main functions for each core

Each core can have its own main function where the activities are sorted. All the cores are started with the same main function and with switch-statement, the cores and their functions are distinguished. Therefore, it is decided within the software which core is going to be responsible for the execution of which part of the application.

LED blinking program on single core

```
int main(void)
{
// user-defined SystemClock of 200MHz
#ifdef USE_200MHz
        Sys_WaitMicroSecond(100);
        SYSTEM_Init();
        Sys_WaitMicroSecond(100);
#else
    SCU_CCUCON1.B.STMDIV = 1;
    SCU_CCUCON1.B.UP = 1;
#endif
    // Initialize on-board LEDs
        LEDCTL_Init();

    // Initialize external user button
        Button_Init();

    while(1)
    {
        if(Button_Pressed())
        {
            // LED turns on if button pressed
            LEDCTL_On(1);
        }
        else
        {
            //LED off while button is not pressed
            LEDCTL_Off(1);
        }

        // Wait 100 ms
            Sys_WaitMicroSecond(100000);
    }

    return 0;
}
```

The programs are designed to meet the desired requirements and to check the expected behaviors. For simple input signal switches have been used and to see the desired output, onboard LEDs are controlled to see the responses. Afterward, several other peripherals were considered to be simulated using the simulator. The peripherals are needed to be handled carefully. Unexpected behavior can appear because of concurrent access of multiple peripherals.

7.2 Simulation using UDE with TSIM

While writing and running a program for the TriCore was a smooth and successful procedure, simulating all the three cores of TriCore is not an easy task to do. There is a very limited option for the simulator of a specific processor family. In this case, a simulator for all these cores is not available on the market yet. The manufacturer of the AURIX, Infineon Technologies offers a simulator itself that is called TriCore SIMulator (TSIM) which can simulate only one core at a time. Which is a big drawback while simulating a program that is aimed for a multicore processor. The program is needed to be modified to use it into the simulator. Writing a program for a single core easier compared to a multicore processor. There is no need for any planning to where variables are linked.

Simulation in UDE requires the Universal Access Device to be connected with the host pc. This makes it recognize the tool and simulator interface. The integrated development environment in UDE makes it easier to develop the program that is intended to be simulated. After development, compilation and successful run of the application program, a simulation project is created. The compiled hex file is called & loaded from this project.

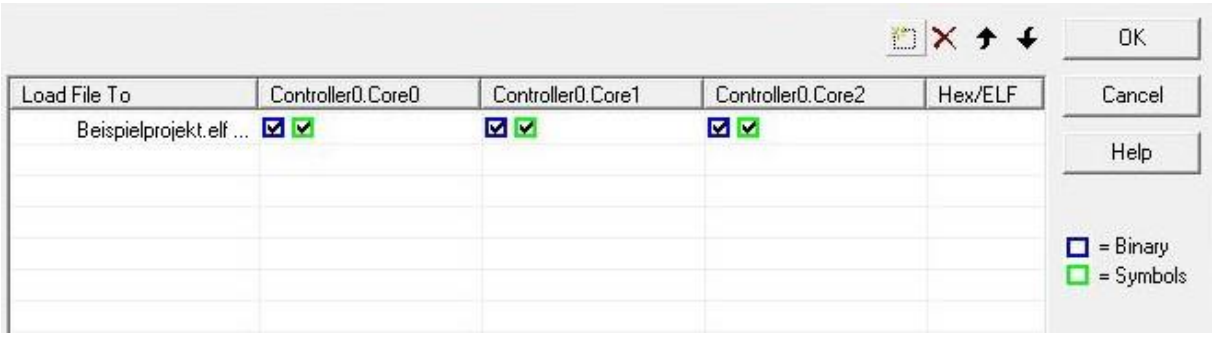

Figure 7.4: Program loader

After loading the program, the project starts it with a default configuration that has Core0 of the program as activated. All the functions of the program are designed to be run on this specific core only.

TSIM simulator is specifically designed for the performance analysis and the implementation of the Instruction Set Architecture (ISA) of AURIX TriCore. [] The simulator usually comes integrated with a source level debugger from a third-party provider. With this, it is possible to both simulate and debug the programs that are intended for the controller.

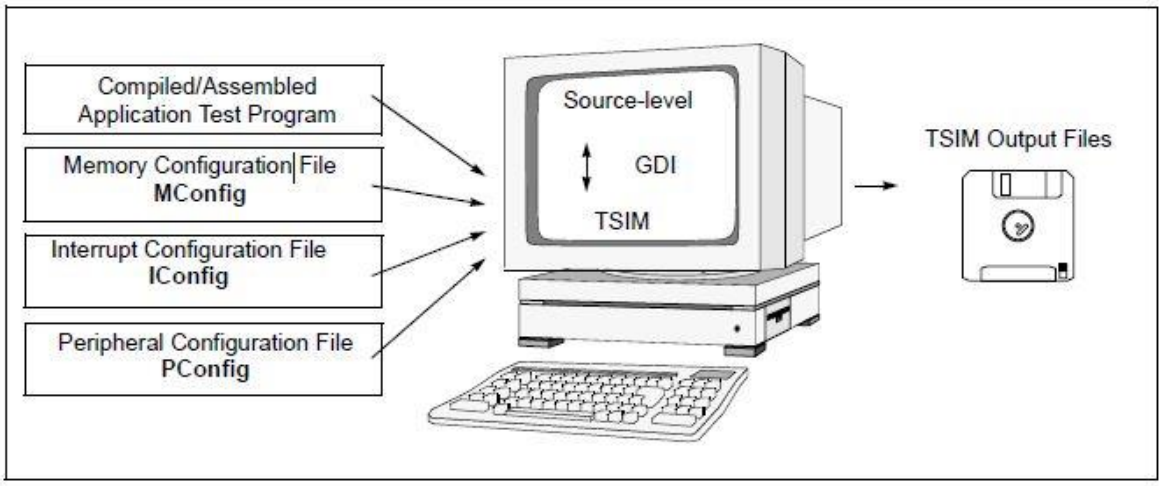

Figure 7.5: TSIM Simulation Environment [33]

The simulator features a re-programmable environment for configuring the specific versions of a TriCore-based microcontroller with an interrupt mechanism, memory, and peripheral address mapping. Thus, makes it very efficient for debugging, performance analysis, functional analysis and trade-off analysis for TriCore based application.

TSim makes the simulation process very flexible and convenient with the feature to customization of several configurations such as Memory configuration (MConfig), interrupt configuration file (IConfig), peripheral configuration (PConfig) and device configuration (DConfig). The configurations can be added as an external configuration file during the simulation.

The simulation for the purpose of this thesis work was carried out at first with the default configurations from the simulator to test out the environment. Afterward, specific configuration files were loaded that were required for the TC275 controller.

```
1  BOOTADDRESS 0xD6000000
2  EXITADDRESS 0xD6000f00
3  CACHE DATA 1 4 1
4  CACHE CODE 1 2 2
5  SCRATCH DATA0 0xD0000000 0xD0007fff 1
6  SCRATCH DATA1 0xD2000000 0xD2007fff 1
7  SCRATCH CODE0 0xD4000000 0xD4007fff 1
8  SCRATCH CODE1 0xD6000000 0xD6007fff 1
9  R 0xC0000000 0xC0001fff 3
10 R 0x80000000 0x8000ffff 3
11 R 0xA8000000 0xA000ffff 7
12 __IOREAD 0xa0000000
13 __IOWRITE 0xa0000004
14 __CYCLES 0xa0000008
15 SFR_BASE_ADDRESS 0xf7e10000
16 DMI_BASE_ADDRESS 0xf87ffc00
17 PMI_BASE_ADDRESS 0xf87ffd00
18 LFI_BASE_ADDRESS 0xf87fff00
19 WDT_BASE_ADDRESS 0xf0000000
20 STM_BASE_ADDRESS 0xf0000300
21 GPTU0_BASE_ADDRESS 0xf0000600
```

Memory configuration (MConfig)

Definitions

BOOTADDRESS – Specifies the address from which the core will start execution.

EXITADDRESS – When the core reaches this address, execution will stop.

CACHE DATA - Specifies data cache enabled, 4 ways set associative, and 1-cycle latency.

CACHE CODE – Specifies code cache enabled, 2 ways set associative, and 2-cycle latency.

SCRATCH DATA - Specifies data scratchpad with an address range

R - Memory address with a range

IOREAD – Specifies the address that when reached by the program counter, the system reads from standard in (STDIN) to data register 5 in the TriCore register file.

IOWRITE – Specifies the address that when reached by the program counter, the system writes to standard out (STDOUT) from data register 5 in the TriCore register file.

CYCLES – Specifies the address that when reached by the program counter

BASE_ADDRESS – specifies the base address for a particular set of registers.

```
1  ONESHOT 100 1 6
2  FIX_INTERVAL 100 200 10 2 6 7
3  JITTER 0 200 10 5 3 6 7 8
4  RANDOM 300 350 4 6 7 8 9
```

Interrupt configuration (IConfig)

ONESHOT – ONESHOT interrupt, starting at a specific cycle for 1 peripheral with an SRN number.
FIX_INTERVAL – FIX_INTERVAL with a start and end cycle number.
JITTER –JITTER interrupt starting at specific cycles and ending at cycle number.
RANDOM – RANDOM interrupt starting at specific cycles and ending at 350 cycles for certain peripherals with 4 SRN numbers.

7.3 Output from the Target Microcontroller

While working with the microcontroller, programs that can control General purpose input-output, onboard LEDs were considered so that the program could be easily transported to the simulator and check out the outcomes. Primarily the outputs were checked using onboard LEDs. The TriBoard comes with 8 onboard LEDs that can be controlled through the input signals e.g. using buttons etc. The expected behaviors were checked using the signal from the switch and patterns of the LEDs.

7.4 Simulation Result

Once the program has been successfully flashed and run on the target board, the compiled program is imported to the simulator as Executable and Linkable Format (ELF). An Elf file contains the hex code from the compiled microcontroller program as well as the debugging information.

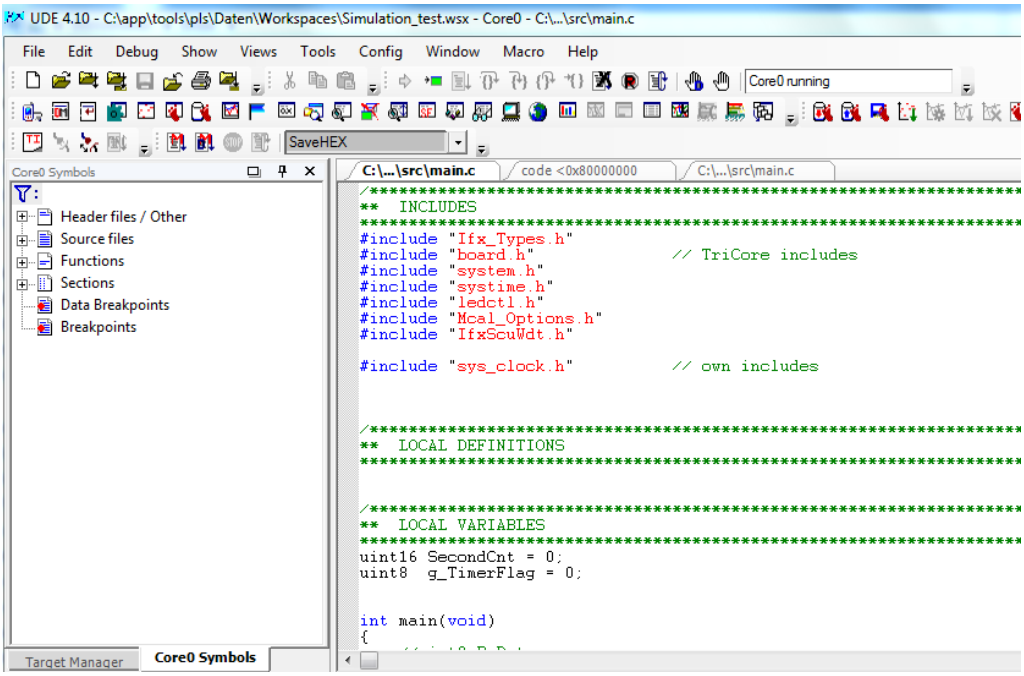

Figure 7.6: UDE Simulator project

After the program is imported to the simulator, it is run on the specific core of microcontroller that the simulator offers. A breakpoint can be set to debug the program both online and offline.

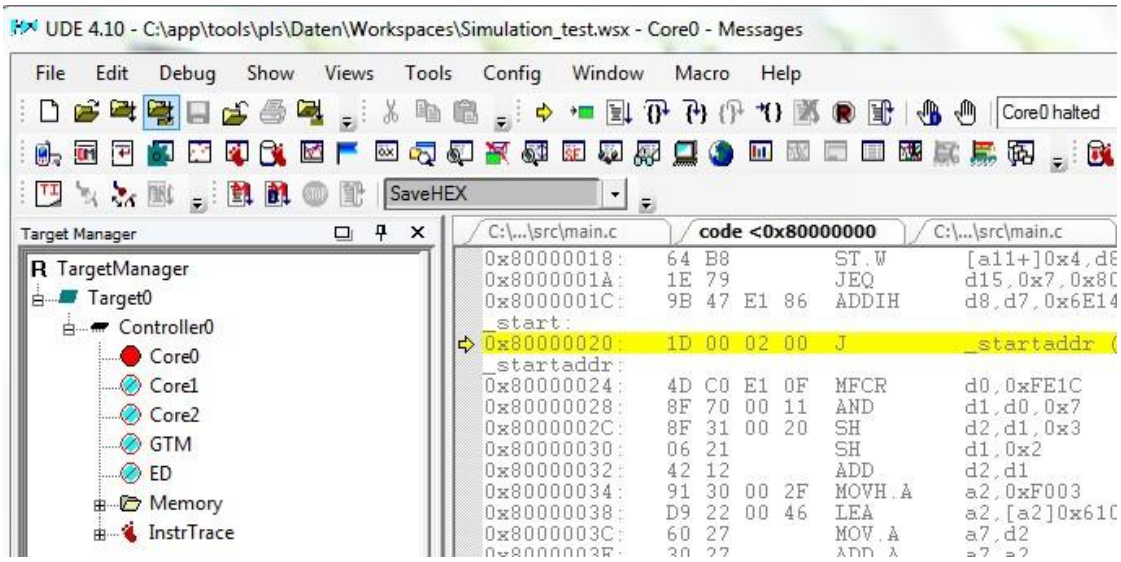

Figure 7.7: Debugging program with breakpoints

While debugging, after execution of each instruction, single step through is possible to check registers, variables, and addresses. Which makes it very useful to find the errors and bugs and improve the program. It is possible to check the registers, variables and addresses again when an interrupt occurs.

After a successful simulation UDE creates an output file as *Sim.out*. The file contains the statistics of the simulation that is just run. This output file can be used to analyze the performance of the simulation.

```
Simulation complete
***************** Simulation statistics *****************
Total number of instructions executed = 564878
Total number of cycles run = 96641
Total number of seconds for execution = 4.604 seconds
total number of interrupts fired=0

No Dcache accesses
No Ccache accesses
Instruction histogram: % (Num of instructions)
MAC: 0% (4)
JMPS: 18% (1304)
LOADS: 19% (1348)
STORES: 15% (1068)
ARITHMETIC: 46% (3210)
SYS: 0% (35)
MISC: 0% (0)
Data Register usage for 32 bit instructions:
DR[00] = 1029 DR[01] = 2201 DR[02] = 148 DR[03] = 113
DR[04] = 109 DR[05] = 33 DR[06] = 2 DR[07] = 0
DR[08] = 56 DR[09] = 22 DR[10] = 10 DR[11] = 7
DR[12] = 3 DR[13] = 0 DR[14] = 470 DR[15] = 1056
Address Register usage for 32 bit instructions:
AR[00] = 3 AR[01] = 3 AR[02] = 433 AR[03] = 2127
AR[04] = 680 AR[05] = 611 AR[06] = 89 AR[07] = 17
AR[08] = 3 AR[09] = 3 AR[10] = 488 AR[11] = 69
AR[12] = 78 AR[13] = 119 AR[14] = 480 AR[15] = 712
Data Register usage for 16 bit instructions:
DR[00] = 717 DR[01] = 2081 DR[02] = 114 DR[03] = 53
DR[04] = 43 DR[05] = 42 DR[06] = 2 DR[07] = 0
DR[08] = 110 DR[09] = 85 DR[10] = 79 DR[11] = 80
DR[12] = 74 DR[13] = 68 DR[14] = 71 DR[15] = 943
Address Register usage for 16 bit instructions:
AR[00] = 0 AR[01] = 0 AR[02] = 588 AR[03] = 1081
AR[04] = 1128 AR[05] = 1075 AR[06] = 97 AR[07] = 3
AR[08] = 0 AR[09] = 0 AR[10] = 422 AR[11] = 68
AR[12] = 121 AR[13] = 164 AR[14] = 88 AR[15] = 386
```

Figure 7.8: Part of a simulation output file

Above figure shows a part of a simulation output file where important statistics of a simulation are shown. Total number instructions executed, number of cycles time required for the run is listed for performance analysis. These performance data can be used to make initial performance comparisons between different implementation approaches for the same processor variant or different processor models. The histogram provides information about individual instructions in several categories.

The usage of data register and address register can be studied here for both 32-bit instructions and 16-bit instructions.

TSIM simulation output File also displays the following important parameters:

- The contents of input configuration files (MConfig, PConfig, IConfig, DConfig) used for this simulation.

```
8    No memory (MConfig) file specified/Error opening: ./MConfig
```

If a memory configuration is not provided or contains errors then, the output file would contain the error message. Thus, the modification can be done on the MConfig file.

- Cache statistics

```
13   TSIM CODE CACHE enabled: 0, size: 8192, line size: 32, ways: 2,
14   access time: 0
15   TSIM DATA CACHE enabled: 0, size: 2048, line size: 16, ways: 2,
16   access time: 0
```

The cache statistics shows the data related to Code Cache and Data Cache and hit-miss ratio.

- Number of instructions executed.
- Number of seconds that the simulation ran.
- Histogram of the instruction mix (total number of instructions and percent of total
  instructions).
- Register file usage and display.

7.5 Performance analysis

Using the simulation output file, the performance of the application code can be tracked. At various stage of the project, measurements are performed on different sections of program code. Doing so, a software implementation ca be tested to determine if the functionality and the performance of the software meets the specific requirements. The performance evaluation is done in an iterative process. The simulation information is interpreted and analyzed at each phase of the total iterative process. The whole cycle can be repeated as many times as necessary to compare outcomes from different configurations of the TriCore.

8 Integration with SIL Environment

The final stage of the proposed prototype environment is to run the microcontroller simulation in the Software-in-the-Loop testing environment. In this case running the TriCore simulation with Softcar. For this purpose, the available techniques that Softcar offers are analyzed and a several approaches has been carried out. The Softcar itself can call and run executables as external process. It is also possible to create a Functional Mockup Unit (FMU) project and load specific DLLs.

8.1 Attempted processes to run the simulator with Softcar

The attempted approaches to run the microcontroller simulation in SIL are described here. The approaches were specific to the simulation tools that are used and the Softcar.

8.1.1 Softcar FMU Loader with FMI

Functional Mock-up Interface (FMI) is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code. It is implemented using a model of the system that has to be simulated, an executable which is written in C which is named a Functional Mock-up Unit (FMU) [53] [54].

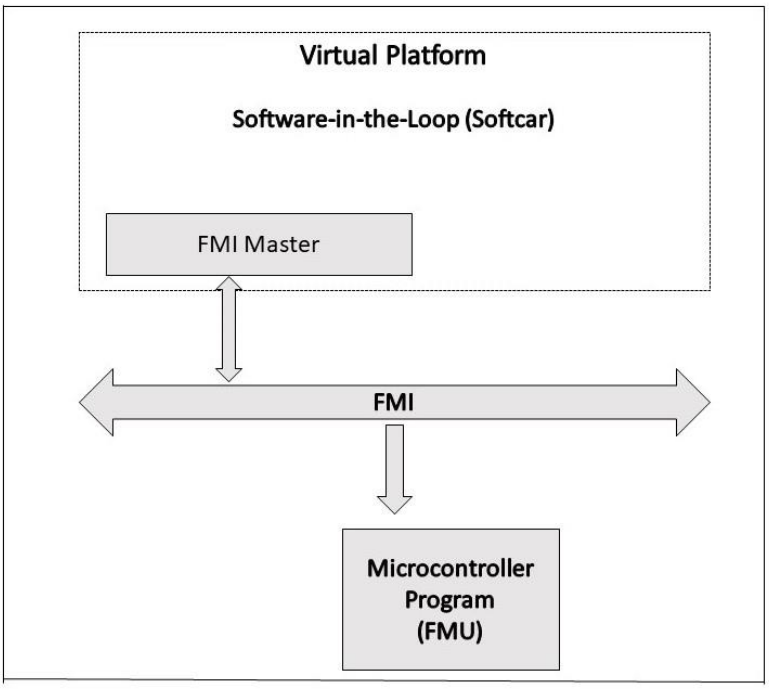

Figure 8.1: Virtual platform interfacing with FMU [52]

It is possible to create an FMU-Loader project in Softcar to load models and load Dynamic-link library (DLL) of a specific program and call the necessary functions. Here, Softcar as a virtual platform can use a configuration to make use of FMI Master. FMI Master works as a communication module between the platform and the FMI interface. The FMI interface then can make use of the microcontroller program as a Functional-Mockup Unit (FMU).

Pseudo code of FMU Loader program

```
Function Main
    Pass In: nothing
    Call:load FMU function
        Load DLL

    IF a DLL loading fails
        Print "DLL Load fails"
        EXIT the program

    Instantiate: FMU function

    IF instantiation fails
        Print "Create Instance failed"
        EXIT the program
    Pass Out: nothing
Endfunction

Function load FMU
    Pass In: nothing
    Load: DLL
    IF DLL exists
        Call: DLL functions
    ELSE
        PRINT "DLL load fails"
    Pass Out: value zero to the operating system
Endfunction
```

Problem with FMU Loader Project:

The FMU loader requires to import the TSIM.DLL that contains all the necessary function to simulate TriCore. Unfortunately, the functions inside the DLL are compiled in a way so that they cannot be called from any programs. There lies also the license issue. The simulator can only be used along with licensed components it requires.

8.1.2 Creating an executable to use TSIM.dll

As the Softcar tool can use an executable directly to run the simulation, it is also a possibility to use the microcontroller simulation by creating an executable that can call the functions necessary for the simulator.

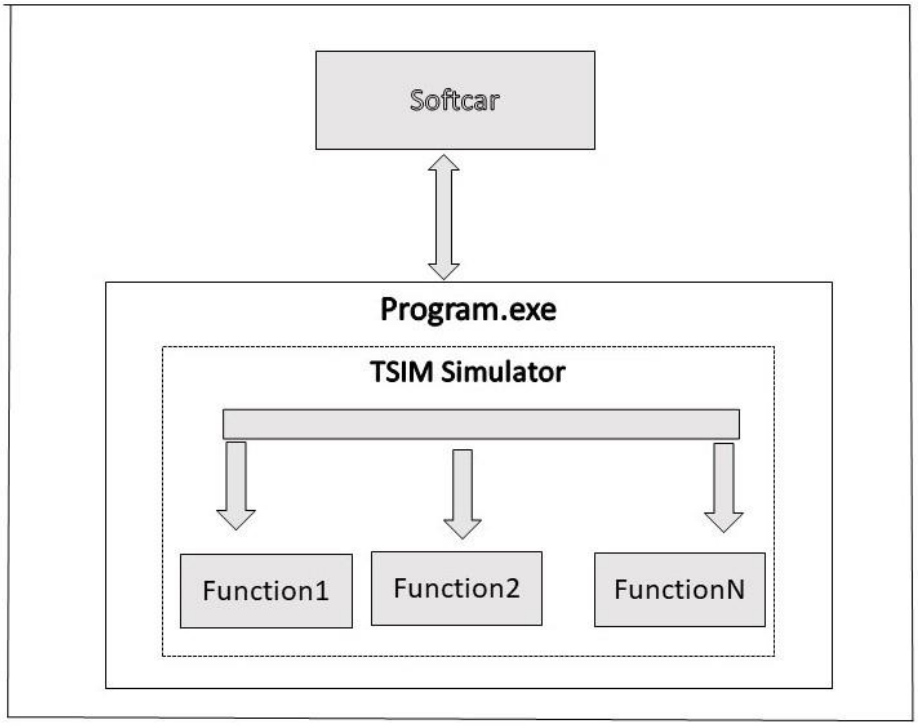

Figure 8.2: Executable to call & run the Simulator

For this purpose, an executable program is written, and attempts were made to run the TSim simulator as an external process in Softcar.

The process of running the TSim simulator standalone with this exe was also unsuccessful. The reason is this that, the TSim simulator is dependent on the third-party debuggers and cannot run without them. There are certain dependencies that are required in extension to the simulator itself.

8.1.3 Use of Lauterbach API

Lauterbach Trace32 debugger provides an API called Peripheral Simulation Model (PSM) that can be used to communicate, retrieve and send data from / to the debugger to / from peripherals and external systems. It is a software overlay for memory area occupied by peripheral module. This API takes over the role of an interaction between the simulator and other modules such as SIL systems. Hence, it gives a possibility to use the Lauterbach debugger with Softcar process.

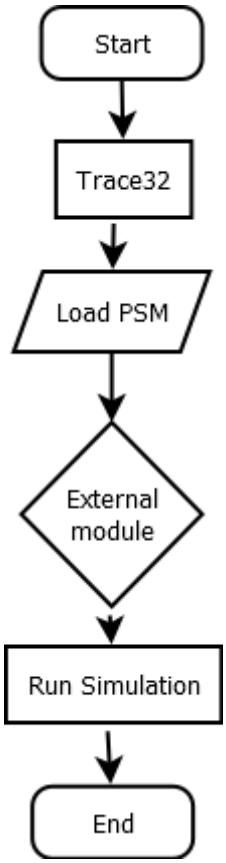

Figure 8.3: Process of using Lauterbach PSM API

Lauterbach Trace32 simulator can simulate variety of processor models which includes TriCore microcontroller processor. And with the Peripheral Simulation Model, Lauterbach simulator can make it possible to simulate peripherals and run with external modules to make the best use out of the microcontroller simulation [42].

Even though, this method has a potential possibility of running the complete simulation of TriCore, it deviates from the primary structure of the imagined prototype environment. Because, the prototype was imagined so that the TSim TriCore simulator from Infineon Technologies could be used. But in this case, the simulator form the Lauterbach has to be used in order to run the complete process with Softcar.

As a last theoretical approach to find a solution, writing a plugin for the debugger is thought to be a possible way. Some of the source level debuggers could function using an external plugin to be able to run in the SIL environment. Writing this plugin would take details study of the requirements and functionalities of the debugger and

of the Softcar tool. So that required components and a protocol of the plugin could be designed and implemented accordingly.

8.2 Multicore Approach

8.2.1 Problem in Multicore Simulation

TSim instruction set simulator cannot simulate multiple cores at a time. Therefore, a simulation of only one core was done within the scope of this work.

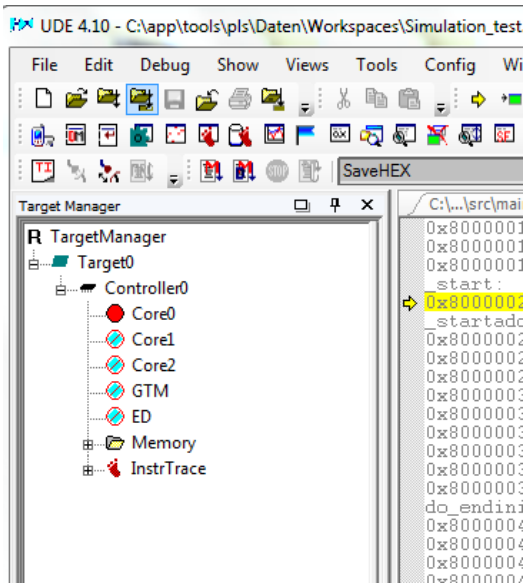

Figure 8.4: Single Core simulation in TSim

As it is depicted in the above image, only Core0 of the controller is active in the project. The individual Ram and the memory are used while running program into the simulator. As the primary goal is to find a solution for the simulation of multiple cores of the process, a thorough theoretical study has been done and a few hypothetical solutions to this problem has been presented afterwards.

In this case two different points of view is taken into consideration. One is from the aspect of the Simulator with a debugger, and the second one is the current technologies and methods that are into consideration for doing the multicore simulation.

8.2.2 Possible ways in Simulator

Multiple target controllers

In a TriCore project, the simulator takes all the processor cores and the first processor cores is in active states while the others are kept inactive. It is established that, the simulator can work with Core0 only. But if there is a way to choose which cores of these three would stay active and the program would run on that specific core then, it would give a bit of liberty.

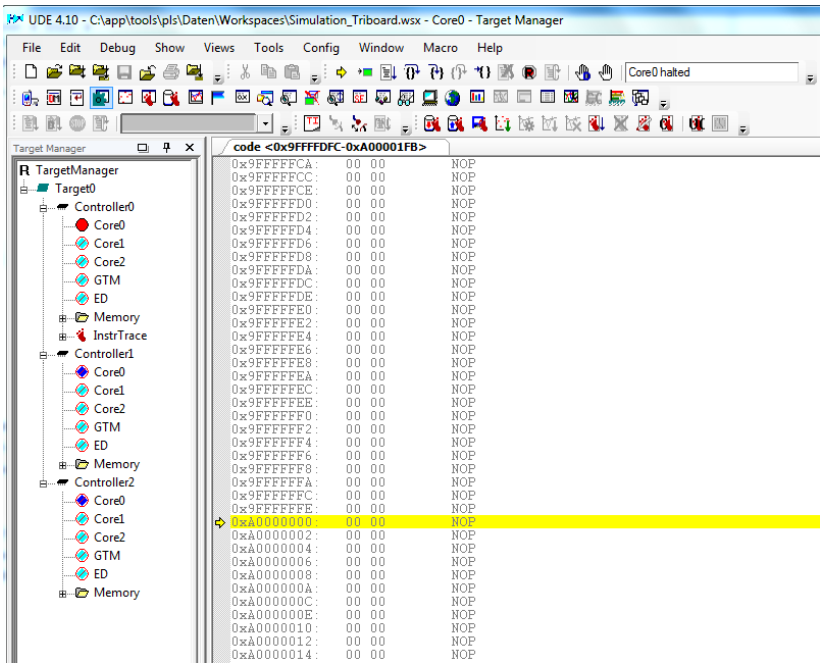

Figure 8.5: Multiple target controller in a program

It is possible to have multiple target controller in a single project. If it is possible to choose the individual cores to run, then theoretically, each target could select separate cores to build a complete project with all three cores.

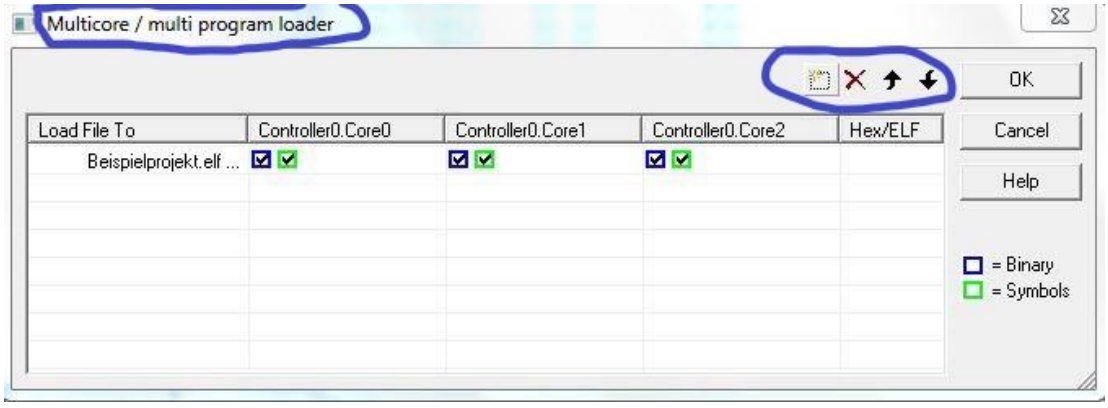

Figure 8.6: Multi program loader

The simulator can load multiple ELF programs as well into the project. This makes it possible to have multiple functions which are intended for different cores. This is also a potential possibility to do a simulation of separate multiple cores.
Several instances of the simulator

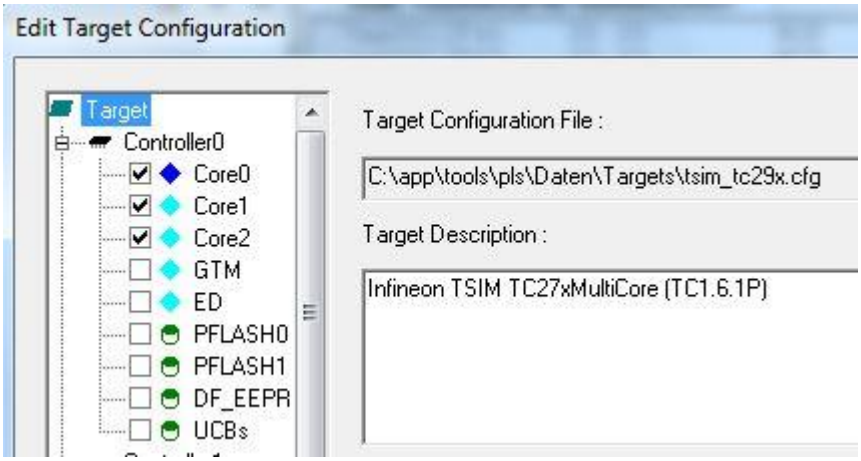

Figure 8.7: Selectable cores in a project

Multiple instances of the Simulator

It is possible to run multiple instances of the simulator on one host one host PC. The instances could be used to a combined complete project if it is possible to establish a proper communication protocol and memory sharing mechanism.

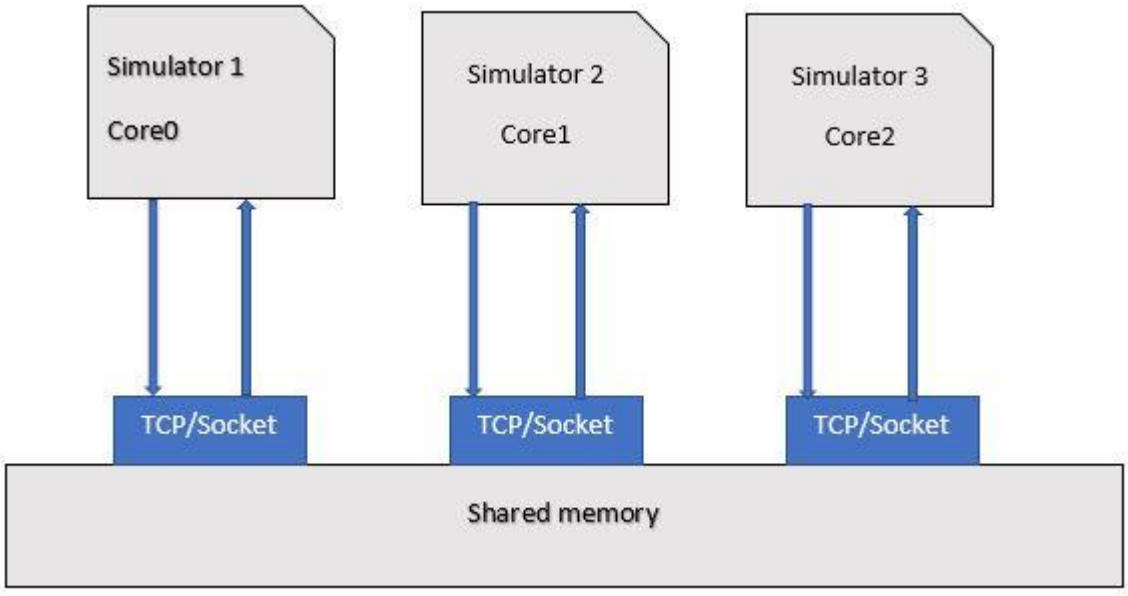

Figure 8.8: Multiple instances of the simulator

The different parts of the program can be assigned to different cores of the instances in this case. As in reality, it can be decided which functions run on which cores, it could be a possibility to assign individual functions to be run on the different instances of the simulator.

With the significant benefits of using the microprocessor simulation, numerous scientific researches are currently in progress to find a viable process to simulate a multicore processor.

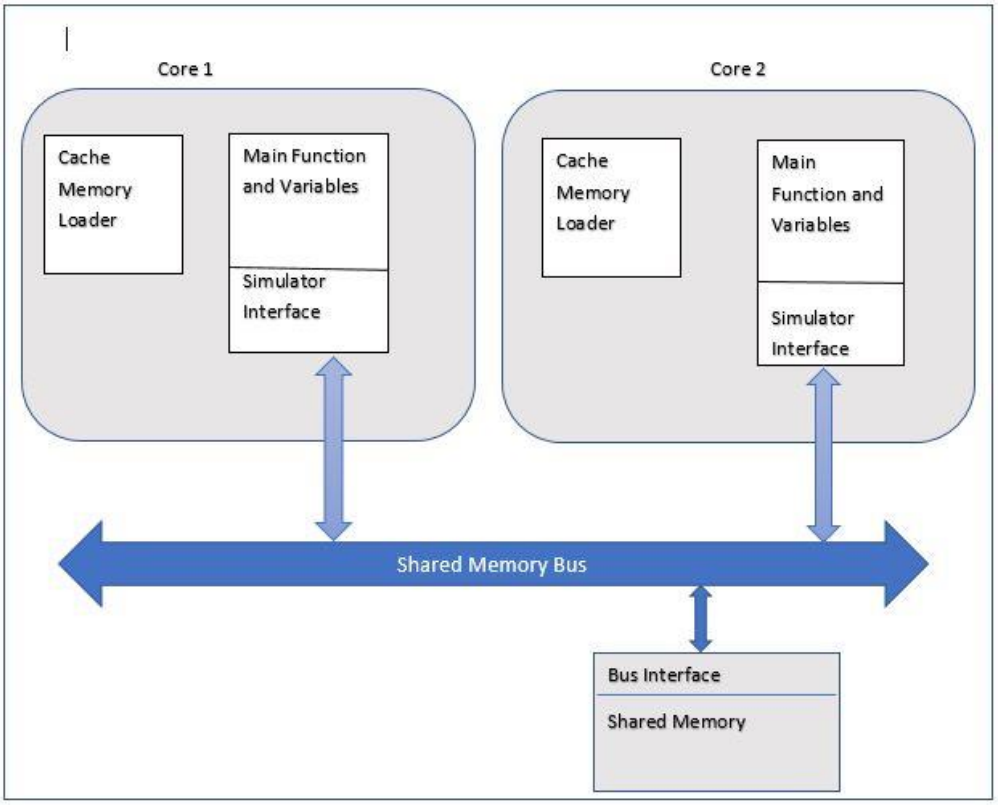

Figure 8.9: Simulation of a dual-core processor [46]

As the figure depicts, the simulation of multicore mostly involves interfacing each core with a shared Memory bus to communicate with each other. A shared memory handles the simultaneous simulation results from each core and transfer to the host pc interface.

8.2.3 Using Synopsis VDKs

Synopsys, Inc provides Virtualizer development kits (VDK) for several automotive software solutions. VDK is a software development kit that uses a fast & virtual prototype as the embedded target and delivers advanced test and debug functionality

Using virtual hardware ECUs [55]. The VDKs support multiple processor families along with several versions of AURIX TriCore family. They are capable of providing virtual ECUs with full multicore support and with a lot of the important peripherals. Which makes it a good choice as tool when it comes to software development and testing for multicore architecture [56].

Virtual prototyping of ECUs with multicore processors has led to various solutions. The most prominent benefit of the VDK is Multicore software development. Besides, Functional safety testing, Regression testing, Vehicle Electrical and Electronic (E/E) architecture and testing, ADAS software and algorithm development, System integration and test using virtual Hardware-in-the-Loop (vHIL) are the major benefits.

9 Conclusion and Future Work

9.1 Conclusion

In this Thesis work, the simulation of an AURIX microcontroller program is carried out using an instruction simulator. The primary target of simulating the multicore was not possible due to unavailability of a suitable simulator. The programs for the microcontroller was written using the programming language C and compiled to a hex file to use it into the simulator.

A thorough market research has been done to find a suitable simulator for AURIX microcontrollers. During the process, the detailed features and functionalities of each simulator were studied and expected. For many of the simulators, the manufacturers and vendors were contacted to find out details about them. If some of the expected criteria were met then, a demo of the tool was installed to find detail functionalities, learn about the user interface, debugging capabilities and other required features. After making a list of possible simulators, the primary criteria were investigated. Which is the ability to simulate the multicore.

The search for the requirement was intended for a specific AURIX model. For this reason, only a shortlisted number of simulators were taken into consideration. As all these simulator tools use TSim simulator, none of them are capable of simulating more than one core. Afterwards, the simulation work and the project were intended to go on with single core simulation.

After a decision on the simulator, a prototype environment is virtualized. For this, a detailed working principle of the SIL platform (Softcar) had to be studied to understand the possible ways to implement the prototype environment.

As a part of it, some application program was written and flashed into the microcontroller to test the functionalities on the physical target-board. With the expected behavior from the hardware, the same compiled file (hex) was then taken into the simulator. The program was then run and debugged into the simulator. The completion of the simulation phase led to the next step of realizing the prototype environment with necessary tools and communication mechanisms.

Several hypotheses are made in order to run this simulation in the SIL environment. Afterward, attempts were made to implement each of the hypothetical methods. The

possible advantages and drawbacks were observed keenly to determine if the method is acceptable or successful. Few Softcar projects were tried out to implement the methods.

Lastly, several researches and works were studied in order to find a theoretical way to do the multicore simulation. Because a multicore simulation with the required peripherals could largely benefit the development process.

9.2 Future Work

This thesis work was done according to the scope of necessary tools e.g. simulators, that are in the market available. To further improve within this goal some different and extended actions could be taken into consideration.

Firstly, the need for multicore simulation is a crucial part of this project. Therefore, a proper tool that can simulate three cores simultaneously is a first thing to do. Even though TSim simulator gives some specific benefits for AURIX simulation, the inability of multicore simulation is a negative point in this case. Therefore, a different simulator category has to be searched, that is not specific to on AURIX simulation, but has the ability to simulate a multicore embedded processor. There are some tools such tools in the market but that are said to be a little more expensive as the process of multicore simulation is a complex procedure.

If a decision can be made on such a tool, the second step would be to pay attention with focus to implement a communication protocol between the simulation tool and SIL environment. The simulation of the microcontroller is supposed to be run in a loop until it is stopped from the SIL. For that the SIL has to have explicit control over the tool while running in the loop.

Alternatively, a different platform other than Softcar can be thought as a substitution. There are some choices for this in the market such as- Silver from QTronic, Synopsis VDK and Isolare-Eve from Etas. These tools have the capability of rapid prototyping using virtual ECU hardware. The Virtualizer Development Kit from Synopsis should be the first choice among these tools. As it has the full potential to simulate multicore processor as well as the capability for development and testing of Software for multicore platforms.

References

[1] C. Ebert and C. Jones, “Embedded Software: Facts, Figures, and Future,” *Computer*, vol. 42, no. 4, pp. 42–52, Apr. 2009.

[2] E. George and M. Pecht, “Tin whisker analysis of an automotive engine control unit,” *Microelectronics Reliability*, vol. 54, no. 1, pp. 214–219, Jan. 2014.

[3] M. Kaiser, “Electronic control unit (ECU),” in *Gasoline Engine Management: Systems and Components*, K. Reif, Ed. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 254–259.

[4] S. Tian, Y. Liu, W. Xia, J. Li, and M. Yang, “Advanced ECU software development method for fuel cell systems,” *Tsinghua Science and Technology*, vol. 10, no. 5, pp. 610–617, Oct. 2005.

[5] Z. Sun and K. Hebbale, “Challenges and opportunities in automotive transmission control,” in *Proceedings of the 2005, American Control Conference, 2005.*, 2005, pp. 3284–3289 vol. 5.

[6] C. Vong and P. Wong, “Case-based adaptation for automotive engine electronic control unit calibration,” *Expert Systems with Applications*, vol. 37, no. 4, pp. 3184–3194, Apr. 2010.

[7] S. Jeong and W. J. Lee, “An automated testing method for AUTOSAR software components based on SiL simulation,” in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 278–283.

[8] J. Mauss, “Chip simulation used to run automotive software on PC,” p. 8, 2014.

[9] V. Tuominen, H. Reponen, A. Kulmala, S. Lu, and S. Repo, “Real-time hardware- and software-in-the-loop simulation of decentralised distribution network control architecture,” *Transmission Distribution IET Generation*, vol. 11, no. 12, pp. 3057–3064, 2017.

[10] Z. Zhong, B. Wu, X. Chen, and X. Chen, “Automotive powertrain co-simulation with Modelica and Simulink,” in *2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific)*, 2014, pp. 1–5.

[11] K. Hassani and W. S. Lee, “A software-in-the-loop simulation of an intelligent microsatellite within a virtual environment,” in *2013 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, 2013, pp. 31–36.

[12] J. Mauss and M. Simons, “Chip simulation of automotive ECUs,” p. 9.

[13] G. M. Casolino, M. AlizadehTir, A. Andreoli, M. Albanesi, and F. Marignetti, “Software-in-the-loop simulation of a test system for automotive electric drives,” in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, pp. 1882–1887.

[14] E. Chrisofakis, A. Junghanns, C. Kehrer, and A. Rink, “Simulation-based development of automotive control software with Modelica,” 2011, pp. 1–7.

[15] “Virtualization of Heterogeneous Electronic Control Units, Testing and Validating Car2X Communication,” ETAS-PGA, Jul. 2017.

[16] H. Brückmann, J. Strenkert, D. U. Keller, B. Wiesner, and D. A. Junghanns, “Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL,” p. 11.

[17] “Simulator for TriCore, Version 20.10.14. Lauterbach GmbH, Höhenkirchen-Siegertsbrunn,Ba., Germany, 2014, pp. 4-30.” .

[18] A. Junghanns, “Virtual integration of Automotive Hard- and Software with Silver,” p. 6.

[19] D. A. Junghanns, R. Serway, D. T. Liebezeit, and M. Bonin, “Building Virtual ECUs Quickly and Economically.” QTronic GmbH, Mar-2012.

[20] Y. Miyazaki and T. Abe, “Trial of multiple ECU co-simulation and fault injection using virtual ECU,” Jul. 2014.

[21] “ISOLAR-EVE User’s Guide, V3.2.ETAS GmbH, Stuttgart,BW., Germany, 2017, pp. 10-30.” .

[22] P. Dong, “Optimized shift control in automatic transmissions with respect to spontaneity, comfort, and shift loads,” p. 309.

[23] F. Galea, E. Gatt, O. Casha, and I. Grech, “Control Unit for a Continuous Variable Transmission for use in an Electric Car,” in *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, 2010, pp. 247–250.

[24] H. Quanan, S. Jian, and L. Lei, “Research on Rapid Testing Platform for TCU of Automated Manual Transmission,” in *2011 Third International Conference on Measuring Technology and Mechatronics Automation*, 2011, vol. 3, pp. 67–70.

[25] D. Xue, X. Yin, and L. Li, “Software architecture for the ECU of automated manual transmission,” in *The 2nd International Conference on Software Engineering and Data Mining*, 2010, pp. 63–68.

[26] J. Qu and L. Liang, “A Production Rule Based Expert System for Electronic Control Automatic Transmission Fault Diagnosis,” in *2009 International Conference on Information Engineering and Computer Science*, 2009, pp. 1–4.

[27] F. Jauch, and G. Girres, “Developing the Networking of AUtomatic Transmissions.” ZF Friedrichshafen AG, 2013.

[28] H. Naunheimer, P. Fietkau, and G. Lechner, Eds., *Automotive transmissions: fundamentals, selection, design, and application*, 2nd ed. Heidelberg ; New York: Springer, 2011.

[29] W. Yu, N. Li, D. Zhao, and S. Han, “Adaptive Fuzzy Shift Strategy in Automatic Transmission of Construction Vehicles,” in *2006 International Conference on Mechatronics and Automation*, 2006, pp. 1357–1361.

[30] W. Yang, G. Wu, and J. Dang, “Research and Development of Automatic Transmission Electronic Control System,” in *2007 IEEE International Conference on Integration Technology*, 2007, pp. 442–445.

[31] H. Sadjadian, S. Ozgoli, and M. Jalalifar, “Design and implementation of automatic transmission electronic control system for mining trucks,” in *Engineering design and Manufacturing informatization 2011 International Conference on System science*, 2011, vol. 2, pp. 1–4.

[32] Z. Zhong, H. Jing, and Y. Ma, “The coordination work between engine management system and TCU in transmission control,” in *2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)*, 2011, pp. 5357–5360.

[33] D. T. Liebezeit, J. Bräuer, R. Serway, and D. A. Junghanns, “Virtual ECUs for Developing Automotive Transmission Software,” p. 9.

[34] AUTOSAR Web Team, “AUTOSAR Introduction,” Jul-2018. [Online]. Available: https://www.autosar.org/fileadmin/HOW_TO_JOIN/AUTOSAR_Introduction.pdf.

[35] S. Bunzel, “AUTOSAR – the Standardized Software Architecture,” *Informatik Spektrum*, vol. 34, no. 1, pp. 79–83, Feb. 2011.

[36] S. Schmerler, “Autosar — Lasting Way To A Global Standard,” *Auto Tech Rev*, vol. 2, no. 1, pp. 68–70, Jan. 2013.

[37] M. Ramseyer, “Microcontroller and Full system Simulation,” p. 31.

[38] A. del Rio and J. J. R. Andina, “UV151: a simulation tool for teaching/learning the 8051 microcontroller,” in *30th Annual Frontiers in Education Conference.*

Building on A Century of Progress in Engineering Education. Conference Proceedings (IEEE Cat. No.00CH37135), 2000, vol. 2, p. F4E/11-F4E/16 vol.2.

[39] L. Gauthier and A. A. Jerraya, “Cycle-true simulation of the ST10 microcontroller including the core and the peripherals,” in *Proceedings 11th International Workshop on Rapid System Prototyping. RSP 2000. Shortening the Path from Specification to Prototype (Cat. No.PR00668)*, 2000, pp. 60–65.

[40] M. Djukic, N. Cetic, R. Obradovic, and M. Popovic, “An Approach to Instruction Set Compiled Simulator Development Based on a Target Processor C Compiler Back-End Design,” in *2009 First IEEE Eastern European Conference on the Engineering of Computer Based Systems*, 2009, pp. 32–41.

[41] J. Espinosa, C. Hernandez, J. Abella, D. de Andres, and J. C. Ruiz, “Analysis and RTL correlation of instruction set simulators for automotive microcontroller robustness verification,” in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[42] C. Bonivento, M. Cacciari, A. Paoli, and M. Sartini, “Rapid prototyping of automated manufacturing systems by software-in-the-loop simulation,” in *2011 Chinese Control and Decision Conference (CCDC)*, 2011, pp. 3968–3973.

[43] S. Werner, L. Masing, F. Lesniak, and J. Becker, “Software-in-the-Loop simulation of embedded control applications based on Virtual Platforms,” in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[44] “‘Softcar User Guide, V7.004. ZF Friedrichshafen AG, Friedrichshafen,BW., Germany, pp. 5-71.’” .

[45] “AURIX TC27x D-Step User Guide, V2.2 2014. Infineon Technologies AG, Munich,,Bayern, Germany, pp. 5-77.”.” .

[46] “Universal Debug Engine for AURIX, V 4.10.00.07. PLS Development Tools, Lauta, Saxony, germany., 2018, pp. 43-91.” .

[47] “TriBoard TC2X5 hardware manual, V 2.3 2013-05. Infineon Technologies AG, Munich,Ba., Germany, pp. 4-61.”.” .

[48] “Instruction Set Simulator (ISS) User Guide, Edition 2005-01. Infineon Technologies AG, München,., Germany, 2005, pp. 10-21.” .

[49] J. Xu, Y. Zhu, L. Jiang, J. Ni, and K. Zheng, “A Simulator for Multi-Core Processor Micro-Architecture Featuring Inter-Core Communication, Power and Thermal Behavior,” in *2008 International Conference on Embedded Software and Systems Symposia*, 2008, pp. 237–242.

[50] “Steuerung und Kontrolle aller TriCore-Kerne...” [Online]. Available: https://katalog.bibliothek.tu-chemnitz.de/Record/ai-48-TVRfX01UMDYxMjAxMDI2. [Accessed: 26-Jun-2018].

[51] A. B. Abdallah, “Introduction to Multicore Systems On-Chip,” in *Multicore Systems On-Chip: Practical Software/Hardware Design*, Atlantis Press, Paris, 2013, pp. 1–17.

[52] H. V. Caprita and M. Popa, “Multithreaded Peripheral Processor for a Multicore Embedded System,” in *Applied Computational Intelligence in Engineering and Information Technology*, Springer, Berlin, Heidelberg, 2012, pp. 201–212.

[53] C. Stoermer and G. Tibba, “Powertrain co-simulation using AUTOSAR and the Functional Mockup Interface standard,” in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–1.

[54] P. Balda, “Real-time simulator of component models based on Functional Mock-up Interface 2.0,” in *2017 21st International Conference on Process Control (PC)*, 2017, pp. 392–397.

[55] V. Reyes, “Using Virtual Prototypes to Address the Growing Software Complexity in Automotive,” p. 16.

[56] V. Reyes, “Using VDKs for Automotive Systems.pdf.” .

Appendix A

A.1 Microcontroller program

For the purpose of simulation, multiple microcontroller program was tried out. As the TriBoard only had onboard LEDs to be controlled, a simple LED blinking program was a perfect match for both program and peripheral simulation. Following is a LED blinking program that was used.

```
/*****************************************************************************
**      INCLUDES
*****************************************************************************/
// TriCore includes
#include "Ifx_Types.h"
#include "board.h"
#include "system.h"
#include "systime.h"
#include "ledctl.h"
#include "Mcal_Options.h"
#include "IfxScuWdt.h"

// custom includes
#include "sys_clock.h"

/*****************************************************************************
**      LOCAL DEFINITIONS
*****************************************************************************/
void M_TimerISR(void);
void M_AppLED(void);
void LED_heller(uint8 LED);
void LED_dunkler(uint8 LED);

/*****************************************************************************
**      LOCAL VARIABLES
*****************************************************************************/
uint16 SecondCnt = 0;
uint8  g_TimerFlag = 0;
```

```
// main function
int main(void)
{
	uint8 RxData;
	uint32 ErrorNo = 0;

	// Disable watchdog

	IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());

	IfxScuWdt_disableSafetyWatchdog(IfxScuWdt_getSafetyWatchdogPassword(
));

#ifdef USE_200MHz
	// SystemClock = 200MHz
		Sys_WaitMicroSecond(100);
		SYSTEM_Init();
		Sys_WaitMicroSecond(100);
#else
	SCU_CCUCON1.B.STMDIV = 1;
	SCU_CCUCON1.B.UP = 1;
#endif
	// Initialize on-board LEDs
		LEDCTL_Init();

	// Initialize external user button
		Button_Init();

	while(1)
	{
		if(Button_Pressed())
		{
			LEDCTL_On(1);
```

```
                }
                else
                {
                        LEDCTL_Off(1);
                }

                // Wait 100 ms
                        Sys_WaitMicroSecond(100000);
        }

        return 0;
}
```

A.2 Simulation Output

After a simulation using TSIM into the Universal debug engine, a simulation output was generated each time. Which includes all the information and statistics of the simulation. Following is a sample output file.

```
---------------------------------------------------------------
Sim.out
---------------------------------------------------------------
INFINEON TECHNOLOGIES TRICORE ISS
ATTENTION: TSIM is an ARCHITECTURAL MODEL
cycle counts are approximate.

No memory (MConfig) file specified/Error opening: ./MConfig


Default Cache Configuration:

TSIM CODE CACHE enabled: 0, size: 8192, line size: 32, ways: 2,
access time: 0
TSIM DATA CACHE enabled: 0, size: 2048, line size: 16, ways: 2,
access time: 0

Default Memory Configuration:
CODE SCRATCH: DEFAULT: 0xD4000000 - 0xD4007FFF, access_time: 1
```

```
DATA SCRATCH: DEFAULT: 0xD0000000 - 0xD0007FFF, access_time: 1
EXTERNAL MEM: DEFAULT: 0x80000000 - 0xCFFFFFFF, access_time: 7
EXTERNAL MEM: DEFAULT: 0xDF000000 - 0xDFFFFFFF, access_time: 7
EXTERNAL MEM: DEFAULT: 0xE2000000 - 0xE2000FFF, access_time: 7
EXTERNAL MEM: DEFAULT: 0xDE000000 - 0xDEFFFFFF, access_time: 7
EXTERNAL MEM: DEFAULT: 0xF0000000 - 0xF0003FFF, access_time: 7
No interrupt (IConfig) file specified/Error opening: ./IConfig
No peripheral (PConfig) file specified/Error opening: ./PConfig
No device plugin (DConfig) file specified/Error opening:
./DConfig
RESET PC = a0000000
All Files setup for simulation run
.....
Downloading program to Simulator
Reset and initialize Simulator
RESET PC = a0000000
Start Simulation
In ExecSingleStep
Simulation stopped due to PWRDN
In ExecStop
In GdiClose
sim done closing up ....
Simulation complete
********** Simulation statistics **********
Total number of instructions executed = 564878
Total number of cycles run = 96641
Total number of seconds for execution = 4.604 seconds
total number of interrupts fired=0

No Dcache accesses
No Ccache accesses
Instruction histogram: % (Num of instructions)
MAC: 0% (4)
JMPS: 18% (1304)
LOADS: 19% (1348)
STORES: 15% (1068)
ARITHMETIC: 46% (3210)
SYS: 0% (35)
```

MISC: 0% (0)
Data Register usage for 32 bit instructions:
DR[00] = 1029 DR[01] = 2201 DR[02] = 148 DR[03] = 113
DR[04] = 109 DR[05] = 33 DR[06] = 2 DR[07] = 0
DR[08] = 56 DR[09] = 22 DR[10] = 10 DR[11] = 7
DR[12] = 3 DR[13] = 0 DR[14] = 470 DR[15] = 1056
Address Register usage for 32 bit instructions:
AR[00] = 3 AR[01] = 3 AR[02] = 433 AR[03] = 2127
AR[04] = 680 AR[05] = 611 AR[06] = 89 AR[07] = 17
AR[08] = 3 AR[09] = 3 AR[10] = 488 AR[11] = 69
AR[12] = 78 AR[13] = 119 AR[14] = 480 AR[15] = 712
Data Register usage for 16 bit instructions:
DR[00] = 717 DR[01] = 2081 DR[02] = 114 DR[03] = 53
DR[04] = 43 DR[05] = 42 DR[06] = 2 DR[07] = 0
DR[08] = 110 DR[09] = 85 DR[10] = 79 DR[11] = 80
DR[12] = 74 DR[13] = 68 DR[14] = 71 DR[15] = 943
Address Register usage for 16 bit instructions:
AR[00] = 0 AR[01] = 0 AR[02] = 588 AR[03] = 1081
AR[04] = 1128 AR[05] = 1075 AR[06] = 97 AR[07] = 3
AR[08] = 0 AR[09] = 0 AR[10] = 422 AR[11] = 68
AR[12] = 121 AR[13] = 164 AR[14] = 88 AR[15] = 386

Selbstständigkeitserklärung

Add this Document:

https://www.tu-chemnitz.de/studentenservice/zpa/formulare/allgemeineformulare/abschlussarbeit_selbststaendigkeitserklaerung.pdf